Quantum algorithms

Daniel J. Bernstein
University of Illinois at Chicago

"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):
stores many "qubits";
can efficiently perform
"Hadamard gate", "T gate",
"controlled NOT 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";
... "Grover's algorithm"; etc.

m algorithms

. Bernstein

ty of Illinois at Chicago

---

um algorithm"

n algorithm that

um computer can run.

quence of instructions,

ach instruction is

ntum computer's

ed instruction set.

o we know which

tions a quantum

er will support?

Quantum computer type 1 (QC1):
stores many "qubits";
can efficiently perform
"Hadamard gate", "T gate",
"controlled NOT 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";
... "Grover's algorithm"; etc.

Quantum
stores a
efficientl
laws of
with as

This is t
quantum
by 1982
physics

ms

n

is at Chicago

---

nm"

m that

ter can run.

instructions,

ction is

puter's

ion set.

w which

antum

pport?

1

Quantum computer type 1 (QC1):
stores many "qubits";
can efficiently perform
"Hadamard gate", "T gate",
"controlled NOT 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";
... "Grover's algorithm"; etc.

2

Quantum compute
stores a simulated
efficiently simulate
laws of quantum p
with as much accu

This is the origina
quantum compute
by 1982 Feynman
physics with comp

ago

n.

ns,

1

Quantum computer type 1 (QC1):
stores many “qubits”;
can efficiently perform
“Hadamard gate”, “T gate”,
“controlled NOT 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”;
. . . “Grover’s algorithm”; etc.

2

Quantum computer type 2 (
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as de

This is the original concept
quantum computers introdu
by 1982 Feynman “Simulati
physics with computers”.

Quantum computer type 1 (QC1):
stores many "qubits";
can efficiently perform
"Hadamard gate", "T gate",
"controlled NOT 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";
... "Grover's algorithm"; etc.

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):
stores many "qubits";
can efficiently perform
"Hadamard gate", "T gate",
"controlled NOT 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";
... "Grover's algorithm"; etc.

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".

m computer type 1 (QC1):
any "qubits";
iently perform
ard gate", "T gate",
led NOT gate".

these instructions work
ain goal of quantum-
er engineering.

e these instructions
ute "Toffoli gate";
non's algorithm";
or's algorithm";
over's algorithm"; etc.

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".

Quantur
efficientl
that any
can com

er type 1 (QC1):
ts";
form
, "T gate",
gate".

structions work
of quantum-
ering.

tructions
oli gate";
rithm";
thm";
rithm"; etc.

2

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".

3

Quantum compute
efficiently compute
that any physical
can compute effici

(QC1):

,

work

um-

c.

2

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".

3

Quantum computer type 3 (
efficiently computes anythin
that any 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 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 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 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.

m computer type 2 (QC2):
simulated universe;
y simulates the
quantum physics
much accuracy as desired.

he original concept of
n computers introduced
Feynman "Simulating
with computers".

belief: any QC1 is a QC2.
proof: see, e.g.,
rdan–Lee–Preskill
um algorithms for
n field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The stat

Data ("s
an eleme
an eleme

er type 2 (QC2):
universe;
es the
physics
uracy as desired.

l concept of
rs introduced
"Simulating
outers".

y QC1 is a QC2.
e.g.,
Preskill
hms for
ories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The state of an al

Data ("state") sto
an element of {0,
an element of {0,

(QC2):

esired.

of
ced
ng

a QC2.

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The state of an algorithm

Data ("state") stored in n b
an element of $\{0,1\}^n$, viewe
an element of $\{0,1,\ldots,2^n -$

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

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

Quantum computer type 3 (QC3):
efficiently computes anything
that any 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.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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 $|a_q|^2 / \sum_r |a_r|^2$.

m computer type 3 (QC3):
y computes anything
physical computer
pute efficiently.

belief: any QC2 is a QC3.
nt for belief:
sical computer must
he laws of quantum
so a QC2 can efficiently
any physical computer.

belief: any QC3 is a QC1.
nt for belief:
're building a QC1.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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 ex

$(1,0,0,$
"$|0\rangle$" in
Measure

er type 3 (QC3):
es anything
computer
ently.

y QC2 is a QC3.
ef:
outer must
quantum
can efficiently
ical computer.

y QC3 is a QC1.
ef:
g a QC1.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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

$(1,0,0,0,0,0,0,0$
"$|0\rangle$" in standard
Measurement proc

(QC3):
g
a QC3.
ently
uter.
a QC1.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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 st

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

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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.

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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.

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

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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.

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

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

The state of an algorithm

Data ("state") stored in n bits:
an element of $\{0,1\}^n$, viewed as
an element of $\{0,1,\ldots,2^n-1\}$.

State stored in n qubits:
a nonzero element of $\mathbf{C}^{2^n}$.
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.

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

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

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

te of an algorithm

state") stored in n bits:
ent of $\{0,1\}^n$, viewed as
ent of $\{0,1,\ldots,2^n-1\}$.

ored in n qubits:
ro element of $\mathbf{C}^{2^n}$.
ng this vector is tough!

its have state
$\ldots, a_{2^n-1}\rangle$ then
ing the qubits produces
ent of $\{0,1,\ldots,2^n-1\}$
troys the state.
ement produces element q
bability $|a_q|^2/\sum_r |a_r|^2$.

Some examples of 3-qubit states:

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

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

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

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

Fast qua

$(a_0, a_1, a$
$(a_1, a_0, a$
is compl
hence "c

gorithm

ored in n bits:
$1\}^n$, viewed as
$1, \ldots, 2^n - 1\}$.

qubits:
t of $\mathbf{C}^{2^n}$.
ctor is tough!

tate
) then
bits produces
$1, \ldots, 2^n - 1\}$
tate.
duces element q
$_q|^2 / \sum_r |a_r|^2$.

Some examples of 3-qubit states:

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

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

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

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

Fast quantum ope

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_1, a_0, a_3, a_2, a_5, a$
is complementing
hence "complemen

its:
ed as
− 1}.

gh!

ces
− 1}

nent q
$a_r|^2$.

Some examples of 3-qubit states:

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

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

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

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

Fast quantum operations, pa

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ t
$(a_1, a_0, a_3, a_2, a_5, a_4, a_7, a_6)$
is complementing index bit 0
hence "complementing qubi

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.

$(0, 0, 4, 0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
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".

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.

$(0,0,4,0,0,0,8,0) = 4|2\rangle + 8|6\rangle$:
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".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
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 $|a_q|^2 / \sum_r |a_r|^2$.

xamples of 3-qubit states:

$0, 0, 0, 0, 0)$ is
standard notation.
ement produces 0.

$0, 0, 0, 1, 0)$ is
standard notation.
ement produces 6.

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

$0, 0, 0, 8, 0) = 4|2\rangle + 8|6\rangle$:
ement produces
robability 20%,
robability 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".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
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 $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a$
$(a_4, a_5, a$
is "comp
$(q_0, q_1, q$

3-qubit states:

) is
notation.
duces 0.

) is
notation.
duces 6.

$, 0) = -7i|6\rangle$:
duces 6.

$) = 4|2\rangle + 8|6\rangle$:
duces
20%,
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".

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$
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 $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_4, a_5, a_6, a_7, a_0, a$
is "complementing
$(q_0, q_1, q_2) \mapsto (q_0$

tates:

$i|6\rangle$:

$+ 8|6\rangle$:

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":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1$

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":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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

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":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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

Complementing qubit 2
= swapping qubits 0 and 2
- ∘ complementing qubit 0
- ∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

antum operations, part 1

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$a_3, a_2, a_5, a_4, a_7, a_6)$
ementing index bit 0,
complementing qubit 0".

$a_2, a_3, a_4, a_5, a_6, a_7)$
ured as (q_0, q_1, q_2),
ting $q = q_0 + 2q_1 + 4q_2$,
bability $|a_q|^2 / \sum_r |a_r|^2$.

$a_3, a_2, a_5, a_4, a_7, a_6)$
ured as $(q_0 \oplus 1, q_1, q_2)$,
ting $q \oplus 1$,
bability $|a_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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

Complementing qubit 2
= swapping qubits 0 and 2
- ∘ complementing qubit 0
- ∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a$
$(a_0, a_1, a$
is a "rev
"control
$(q_0, q_1, q$

erations, part 1

$a_5, a_6, a_7) \mapsto$

$a_4, a_7, a_6)$

index bit 0,

nting qubit 0".

$a_5, a_6, a_7)$

$_0, q_1, q_2)$,

$q_0 + 2q_1 + 4q_2$,

$_q|^2 / \sum_r |a_r|^2$.

$a_4, a_7, a_6)$

$_0 \oplus 1, q_1, q_2)$,

,

$_q|^2 / \sum_r |a_r|^2$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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

Complementing qubit 2
= swapping qubits 0 and 2
- ∘ complementing qubit 0
- ∘ swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_0, a_1, a_3, a_2, a_4, a$
is a "reversible XO
"controlled NOT g
$(q_0, q_1, q_2) \mapsto (q_0$

art 1

$\longrightarrow$

0,

t 0".

$- 4q_2,$

$a_r|^2.$

$q_2),$

$a_r|^2.$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
is "complementing qubit 2":
$(q_0, q_1, q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

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

Complementing qubit 2
= swapping qubits 0 and 2
- complementing qubit 0
- swapping qubits 0 and 2.

Similarly: swapping qubits i, j.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7)$ ↦
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
is a "reversible XOR gate" =
"controlled NOT gate":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
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
= swapping qubits 0 and 2
- ◦ complementing qubit 0
- ◦ 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":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_4, a_5, a_6, a_7, a_0, a_1, a_2, a_3)$
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
$=$ swapping qubits 0 and 2
- $\circ$ complementing qubit 0
- $\circ$ 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:
$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$
$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$
$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$
$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$
$\mapsto (a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6,$
$a_8, a_9, a_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$
$a_{16}, a_{17}, a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$
$a_{24}, a_{25}, a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_6, a_7, a_0, a_1, a_2, a_3)$

plementing qubit 2":

$q_2) \mapsto (q_0, q_1, q_2 \oplus 1)$.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_6, a_1, a_5, a_3, a_7)$

pping qubits 0 and 2":

$q_2) \mapsto (q_2, q_1, q_0)$.

menting qubit 2

ping qubits 0 and 2

nplementing qubit 0

pping qubits 0 and 2.

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

$(a_0, a_1, a$
$(a_0, a_1, a$
is a "To
"control
$(q_0, q_1, q$

$a_5, a_6, a_7) \mapsto$

$a_1, a_2, a_3)$

g qubit 2":

$, q_1, q_2 \oplus 1)$.

$a_5, a_6, a_7) \mapsto$

$a_5, a_3, a_7)$

ts 0 and 2":

$, q_1, q_0)$.

ubit 2

s 0 and 2

ng qubit 0

its 0 and 2.

g qubits i, j.

8

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

9

$(a_0, a_1, a_2, a_3, a_4, a$
$(a_0, a_1, a_2, a_3, a_4, a$
is a "Toffoli gate"
"controlled contro
$(q_0, q_1, q_2) \mapsto (q_0$

$\longrightarrow$

:

1).

$\longrightarrow$

":

2.

$, j$.

8

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

9

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
is a "Toffoli gate" $=$
"controlled controlled NOT"
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_3, a_2, a_4, a_5, a_7, a_6)$
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":
$(q_0, q_1, q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

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

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

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_3, a_2, a_4, a_5, a_7, a_6)$

versible XOR gate" $=$

lled NOT gate":

$q_2) \mapsto (q_0 \oplus q_1, q_1, q_2)$.

e with more qubits:

$a_2, a_3, a_4, a_5, a_6, a_7,$

$_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$

$a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$

$a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$

$_1, a_3, a_2, a_4, a_5, a_7, a_6,$

$_{11}, a_{10}, a_{12}, a_{13}, a_{15}, a_{14},$

$a_{19}, a_{18}, a_{20}, a_{21}, a_{23}, a_{22},$

$a_{27}, a_{26}, a_{28}, a_{29}, a_{31}, a_{30})$.

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

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

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:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7,$

$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$

$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$

$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$

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

$a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$

$a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$

$a_{24}, a_{25}, a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

Reversib

Say p is

of $\{0, 1,$

General

these fas

to obtai

$(a_0, a_1, .$

$(a_{p^{-1}(0)}$

$a_5, a_6, a_7) \mapsto$

$a_5, a_7, a_6)$

OR gate" $=$

gate":

$\oplus\, q_1, q_1, q_2)$.

e qubits:

$a_5, a_6, a_7,$

$, a_{13}, a_{14}, a_{15},$

$20, a_{21}, a_{22}, a_{23},$

$28, a_{29}, a_{30}, a_{31})$

$_4, a_5, a_7, a_6,$

$, a_{13}, a_{15}, a_{14},$

$20, a_{21}, a_{23}, a_{22},$

$28, a_{29}, a_{31}, a_{30})$.

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

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

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 comput

Say p is a permuta

of $\{0, 1, \ldots, 2^n -$

General strategy t

these fast quantur

to obtain index pe

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

$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \cdot$

$\longmapsto$

$=$

$q_2)$.

a_{15},

$_2, a_{23}$,

$_0, a_{31})$

$_6$,

a_{14},

$_3, a_{22}$,

$_1, a_{30})$.

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
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 operatio
to obtain index permutation
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \cdots, a_{p^{-1}(2^n}$

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
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
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \cdots, a_{p^{-1}(2^n-1)})$:

$(a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$
$(a_0, a_1, a_2, a_3, a_4, a_5, a_7, a_6)$
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
$(a_0, a_1, \ldots, a_{2^n-1}) \mapsto$
$(a_{p^{-1}(0)}, a_{p^{-1}(1)}, \cdots, a_{p^{-1}(2^n-1)})$:

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.

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_3, a_4, a_5, a_7, a_6)$

ffoli gate" $=$

led controlled NOT gate":

$q_2) \mapsto (q_0 \oplus q_1 q_2, q_1, q_2)$.

e with more qubits:

$a_2, a_3, a_4, a_5, a_6, a_7,$

$_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15},$

$a_{18}, a_{19}, a_{20}, a_{21}, a_{22}, a_{23},$

$a_{26}, a_{27}, a_{28}, a_{29}, a_{30}, a_{31})$

$_1, a_2, a_3, a_4, a_5, a_7, a_6,$

$_{10}, a_{11}, a_{12}, a_{13}, a_{15}, a_{14},$

$a_{18}, a_{19}, a_{20}, a_{21}, a_{23}, a_{22},$

$a_{26}, a_{27}, a_{28}, a_{29}, a_{31}, a_{30})$.

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

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

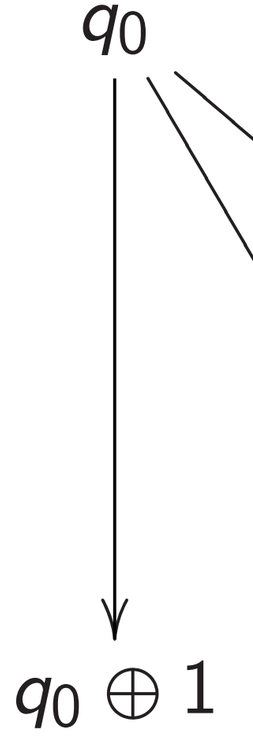
$(a_0, a_1, a$

$(a_7, a_0, a$

permuta

1. Build

to comp



$a_5, a_6, a_7) \mapsto$

$a_5, a_7, a_6)$

$=$

lled NOT gate":

$\oplus\, q_1 q_2, q_1, q_2)$.

e qubits:

$a_5, a_6, a_7,$

$, a_{13}, a_{14}, a_{15},$

$20}, a_{21}, a_{22}, a_{23},$

$28}, a_{29}, a_{30}, a_{31})$

$_4, a_5, a_7, a_6,$

$, a_{13}, a_{15}, a_{14},$

$20}, a_{21}, a_{23}, a_{22},$

$28}, a_{29}, a_{31}, a_{30})$.

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

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 co

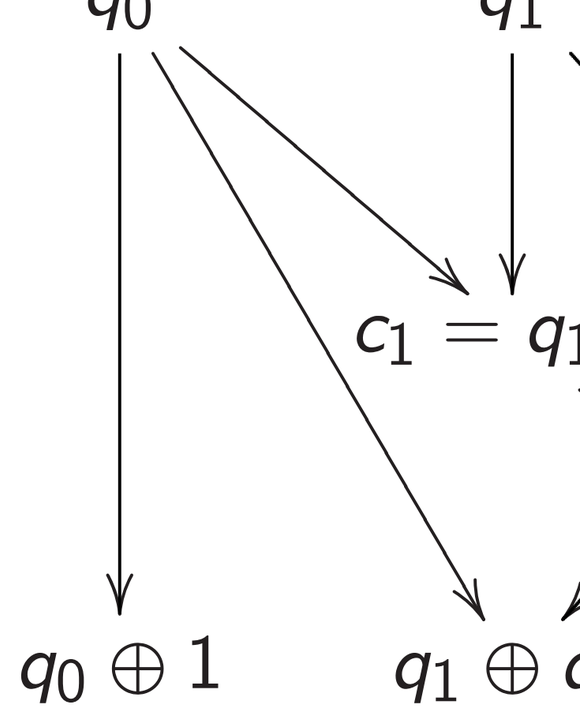
$(a_0, a_1, a_2, a_3, a_4, a$

$(a_7, a_0, a_1, a_2, a_3, a$

permutation $q \mapsto$

1. Build a traditio

to compute $q \mapsto q$



$\longmapsto$

gate":

$_1, q_2)$.

a_{15},

$_2, a_{23}$,

$_0, a_{31})$

$_6$,

a_{14},

$_3, a_{22}$,

$_1, a_{30})$.

Reversible computation

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

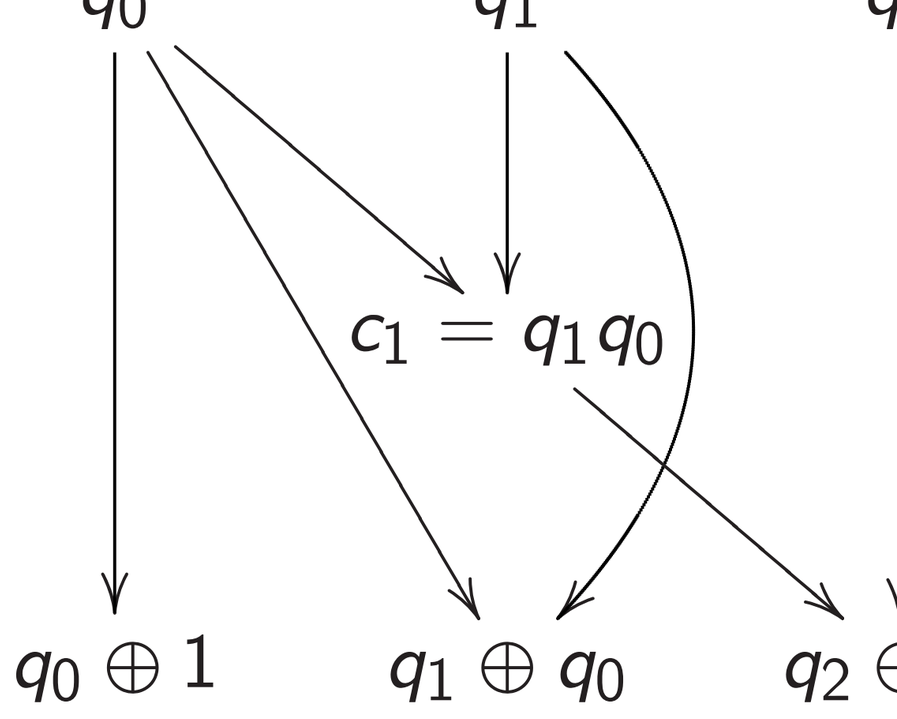
General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

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

1. Build a traditional circuit to compute $q \mapsto q + 1$ mod



Reversible computation

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

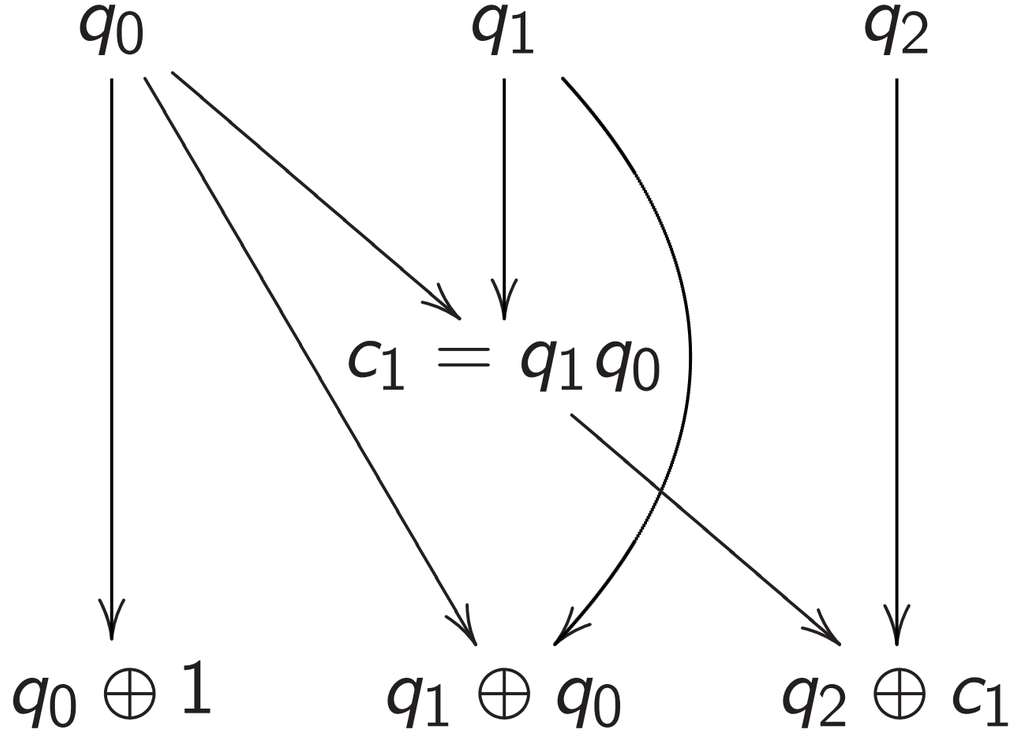
General strategy to compose these fast quantum operations to obtain index permutation $(a_0, a_1, \ldots, a_{2^n-1}) \mapsto (a_{p^{-1}(0)}, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit to compute $q \mapsto q + 1 \bmod 8$.



le computation

a permutation
$\ldots, 2^n - 1\}$.

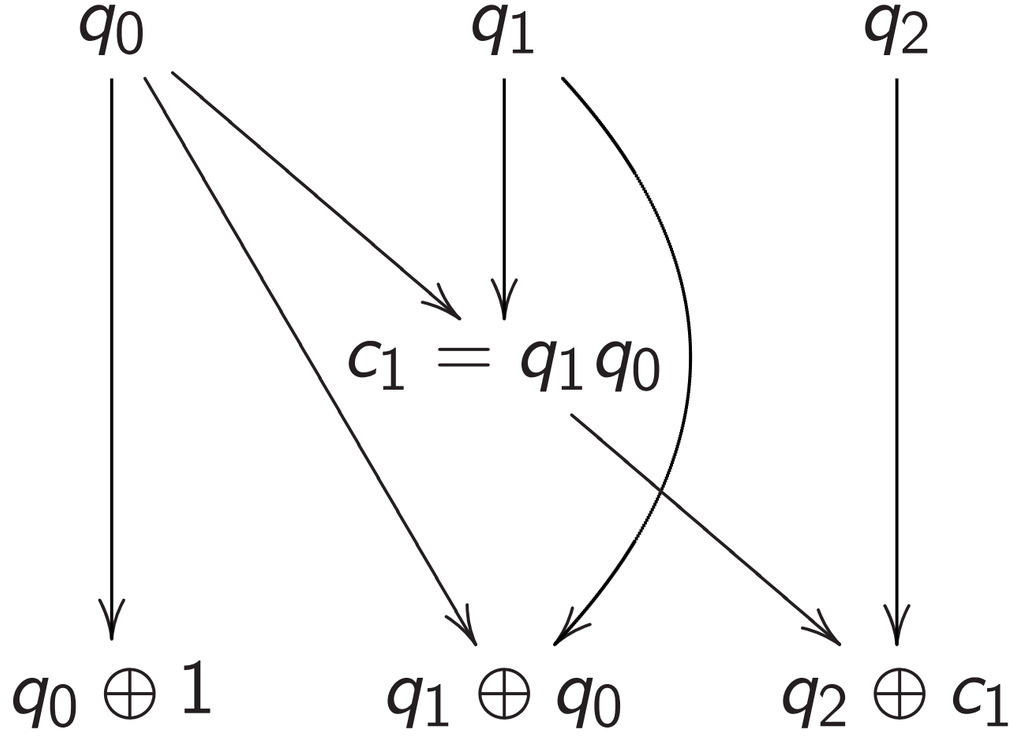
strategy to compose
st quantum operations
n index permutation
$\ldots, a_{2^n-1}) \mapsto$
$, a_{p^{-1}(1)}, \ldots, a_{p^{-1}(2^n-1)})$:

a traditional circuit
ute $j \mapsto p(j)$
OT/XOR/AND gates.

ert into reversible gates:
nvert 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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



2. Conv

Toffoli f
$(a_0, a_1, a$
$(a_0, a_1, a$

ation

ation
1}.

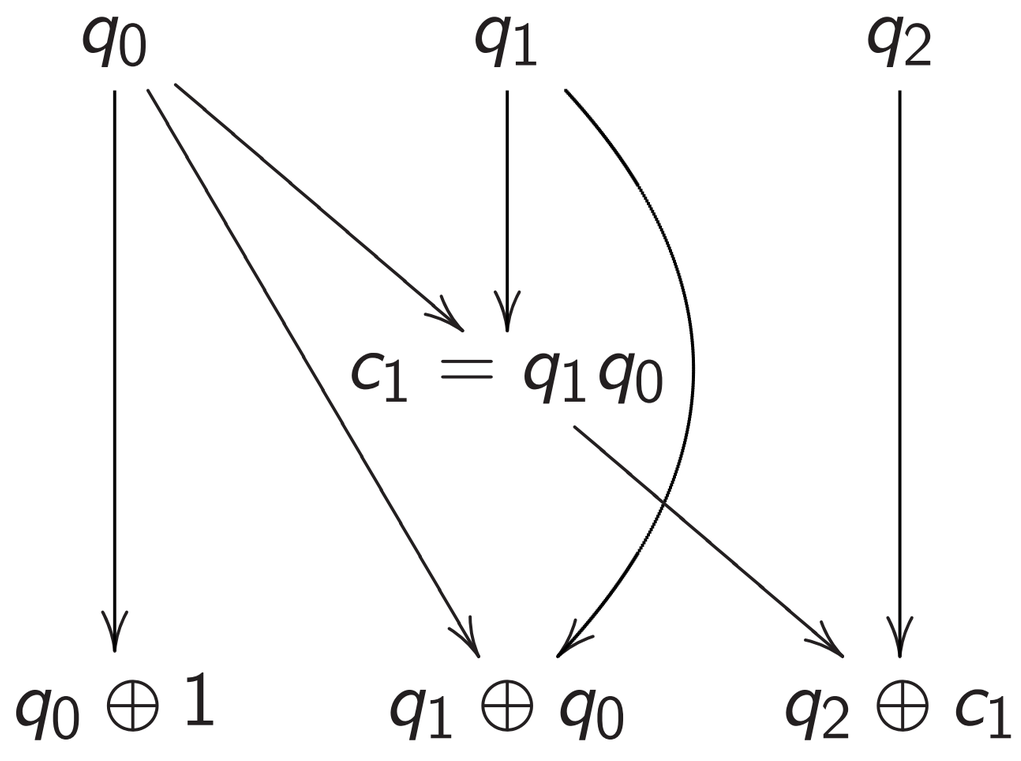
o compose
m operations
ermutation
) ↦
$\ldots, a_{p^{-1}(2^n-1)}$):

nal circuit
(j)
AND gates.

versible gates:
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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.

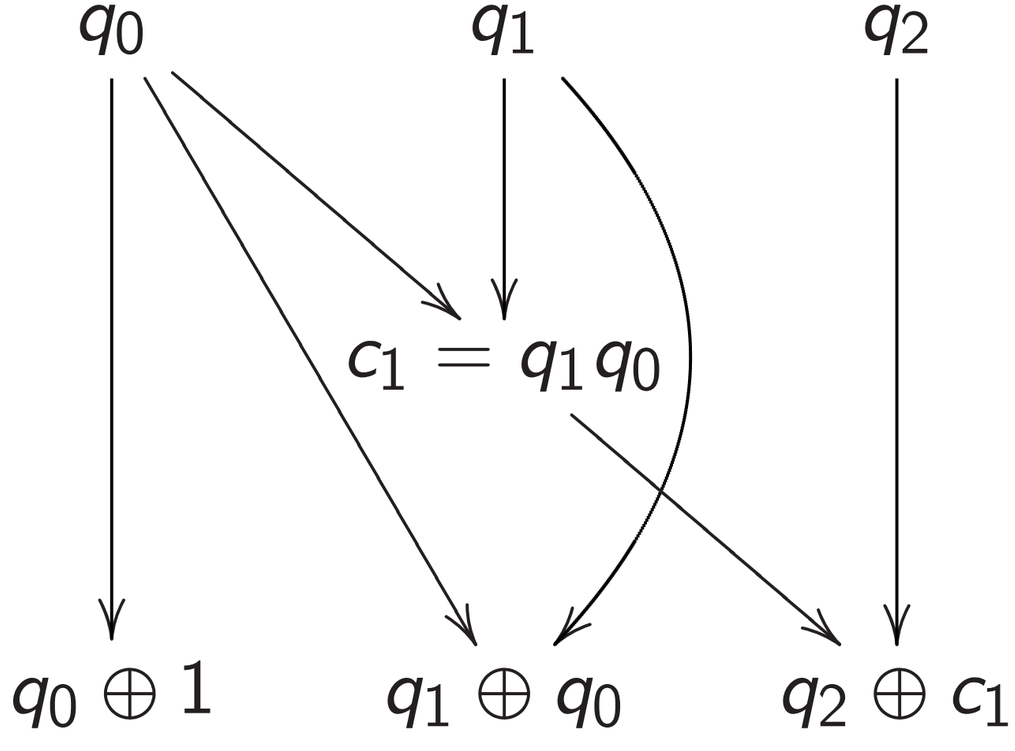


2. Convert into re

Toffoli for $q_2 \leftarrow q$
$(a_0, a_1, a_2, a_3, a_4, a$
$(a_0, a_1, a_2, a_7, a_4, a$

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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.

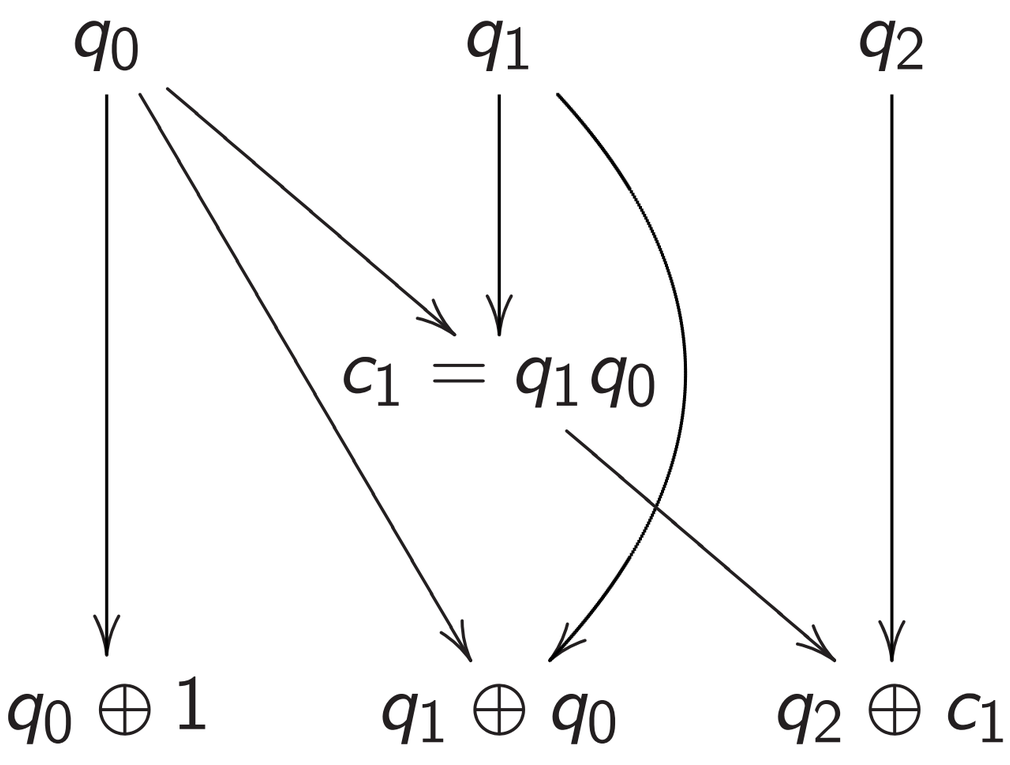


2. Convert into reversible ga

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

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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.

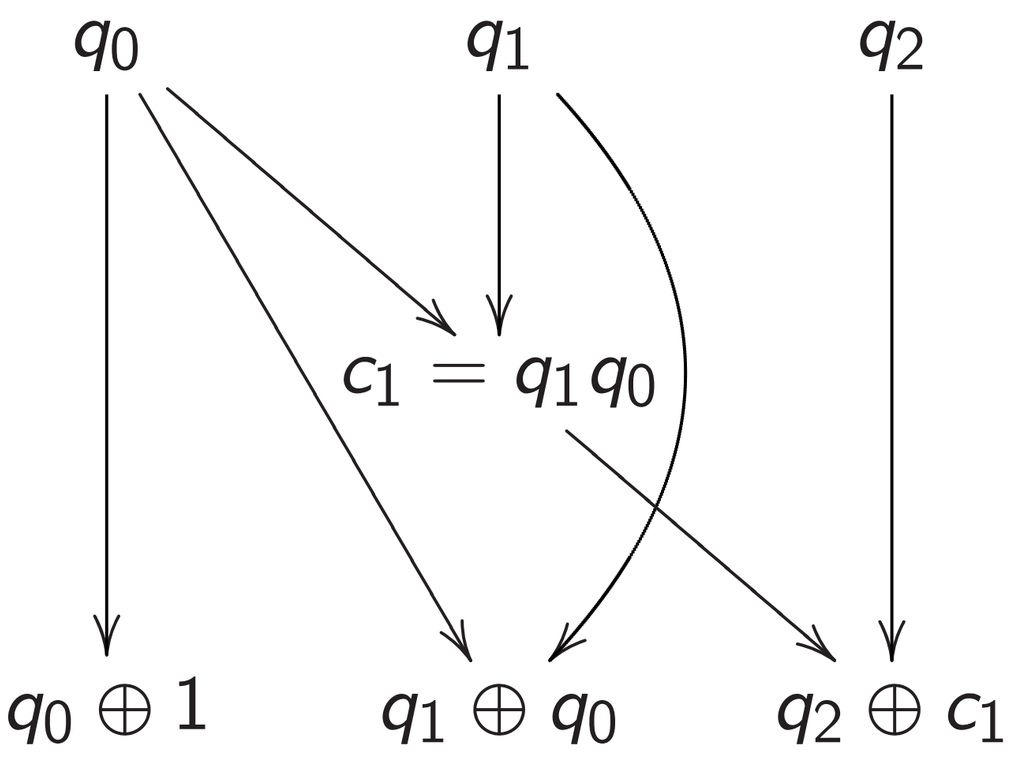


2. Convert into reversible gates.

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

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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



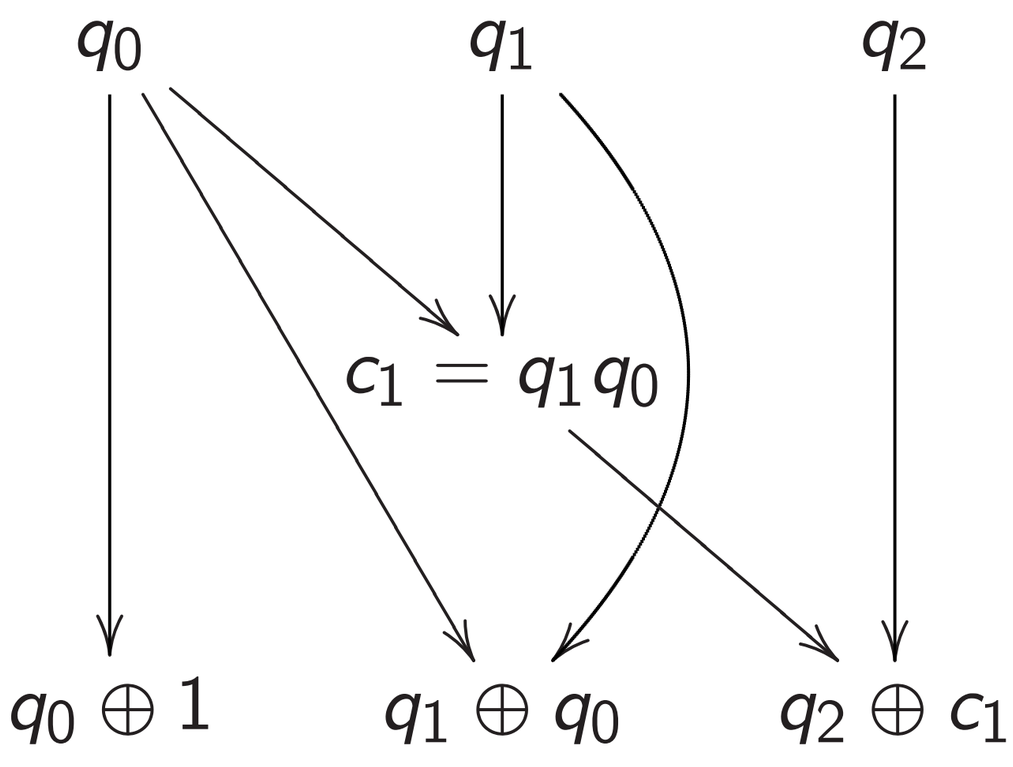
2. Convert into reversible gates.

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

Controlled NOT for $q_1 \leftarrow q_1 \oplus q_0$:
$(a_0, a_1, a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5)$.

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 $q \mapsto q + 1 \bmod 8$.

1. Build a traditional circuit
to compute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

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

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 $q_0 \leftarrow q_0 \oplus 1$:
$(a_0, a_7, a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$
$(a_7, a_0, a_1, a_2, a_3, a_4, a_5, a_6)$.

e: Let's compute

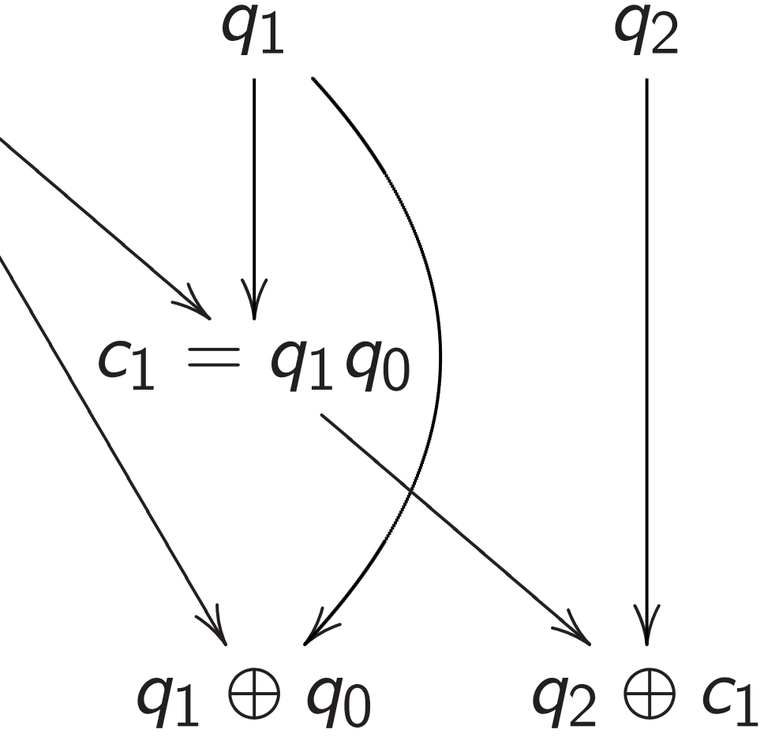
$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_1, a_2, a_3, a_4, a_5, a_6)$;

tion $q \mapsto q + 1 \bmod 8$.

a traditional circuit

ute $q \mapsto q + 1 \bmod 8$.



2. Convert into reversible gates.

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

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

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

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 per

was dece

It didn't

For large

need ma

Really w

mpute

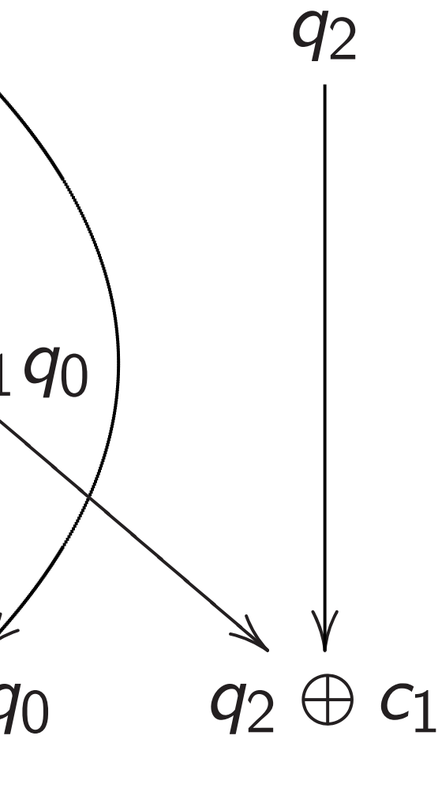
$a_5, a_6, a_7) \mapsto$

$a_4, a_5, a_6)$;

$q + 1 \bmod 8$.

nal circuit

$q + 1 \bmod 8$.



2. Convert into reversible gates.

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

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

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

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

was deceptively ea

It didn't need mar

For large n, most

need many operat

Really want *fast* c

$\longrightarrow$

d 8.

8.

q_2

$\oplus\ c_1$

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 operatio

For large n, most permutati need many operations $\Rightarrow$ slo Really want *fast* circuits.

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.

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.

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.

ert into reversible gates.

or $q_2 \leftarrow q_2 \oplus q_1 q_0$:

$a_2, a_3, a_4, a_5, a_6, a_7) \mapsto$

$a_2, a_7, a_4, a_5, a_6, a_3)$.

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

$a_2, a_7, a_4, a_5, a_6, a_3) \mapsto$

$a_2, a_1, a_4, a_3, a_6, a_5)$.

r $q_0 \leftarrow q_0 \oplus 1$:

$a_2, a_1, a_4, a_3, a_6, a_5) \mapsto$

$a_1, a_2, a_3, a_4, a_5, a_6)$.

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.

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 fro
inputs b
$b_{i+1} = 1$
$b_{i+2} = 1$
...
$b_T = 1 \oplus$
specified

versible gates.

$_2 \oplus q_1 q_0$:

$a_5, a_6, a_7) \mapsto$

$a_5, a_6, a_3)$.

or $q_1 \leftarrow q_1 \oplus q_0$:

$a_5, a_6, a_3) \mapsto$

$a_3, a_6, a_5)$.

$\oplus 1$:

$a_3, a_6, a_5) \mapsto$

$a_4, a_5, a_6)$.

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.

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 cir

inputs $b_1, b_2, \ldots,$

$b_{i+1} = 1 \oplus b_{f(i+1)}$

$b_{i+2} = 1 \oplus b_{f(i+2)}$

...

$b_T = 1 \oplus b_{f(T)} b_{g($

specified outputs.

ates.

$\longrightarrow$

$_1 \oplus q_0$:

$\longrightarrow$

$\longrightarrow$

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.

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 $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

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 $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

Reversible but dirty:
inputs $b_1, b_2, \ldots, b_T$;
$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.
Same outputs if all of
$b_{i+1}, \ldots, b_T$ started as 0.

rmutation example
eptively easy.

need many operations.

e n, most permutations p
ny operations $\Rightarrow$ slow.
want *fast* circuits.

didn't need extra storage:
perated "in place" after
ation $c_1 \leftarrow q_1 q_0$ was
into $q_2 \leftarrow q_2 \oplus c_1$.

circuits aren't in-place.

Start from any circuit:
inputs $b_1, b_2, \ldots, b_i$;
$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;
specified outputs.

Reversible but dirty:
inputs $b_1, b_2, \ldots, b_T$;
$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;
$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;
...
$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.
Same outputs if all of
$b_{i+1}, \ldots, b_T$ started as 0.

Reversib
after fini
set non-
by repea
on non-

Original
(inputs)
(inputs,

Dirty rev
(inputs,
(inputs,

Clean re
(inputs,
(inputs,

example

sy.

ny operations.

permutations p

ions $\Rightarrow$ slow.

ircuits.

d extra storage:

n place" after

$q_1 q_0$ was

$q_2 \oplus c_1$.

en't in-place.

Start from any circuit:

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

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

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

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clea

after finishing dirty

set non-outputs ba

by repeating same

on non-outputs in

Original computat

(inputs) $\mapsto$

(inputs, dirt, outpu

Dirty reversible co

(inputs, zeros, zero

(inputs, dirt, outpu

Clean reversible co

(inputs, zeros, zero

(inputs, zeros, outp

ons.

ons p

ow.

orage:

after

s

ce.

Start from any circuit:

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

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

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

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

Same outputs if all of

$b_{i+1}, \ldots, b_T$ started as 0.

Reversible and clean:

after finishing dirty computa

set non-outputs back to 0,

by repeating same operation

on non-outputs in reverse or

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).

Start from any circuit:

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

$b_{i+1} = 1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} = 1 \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T = 1 \oplus b_{f(T)} b_{g(T)}$;

specified outputs.

Reversible but dirty:

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

$b_{i+1} \leftarrow 1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$b_{i+2} \leftarrow 1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

...

$b_T \leftarrow 1 \oplus b_T \oplus b_{f(T)} b_{g(T)}$.

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).

om any circuit:

$_1, b_2, \ldots, b_i$;

$1 \oplus b_{f(i+1)} b_{g(i+1)}$;

$1 \oplus b_{f(i+2)} b_{g(i+2)}$;

$\oplus b_{f(T)} b_{g(T)}$;

l outputs.

le but dirty:

$_1, b_2, \ldots, b_T$;

$1 \oplus b_{i+1} \oplus b_{f(i+1)} b_{g(i+1)}$;

$1 \oplus b_{i+2} \oplus b_{f(i+2)} b_{g(i+2)}$;

$\oplus b_T \oplus b_{f(T)} b_{g(T)}$.

utputs if all of

$, b_T$ 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 fa
and fast
build fas
$(x, \text{zeros}$

cuit:

b_i;

$)b_{g(i+1)}$;

$)b_{g(i+2)}$;

(T);

y:

b_T;

$\oplus\, b_{f(i+1)}b_{g(i+1)}$;

$\oplus\, b_{f(i+2)}b_{g(i+2)}$;

$_{(T)}b_{g(T)}$.

ll of

ed 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 f
and fast circuit for
build fast reversibl
$(x, \text{zeros}) \mapsto (p(x)$

15

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).

16

Given fast circuit for p
and fast circuit for p^{-1},
build fast reversible circuit f
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

$g(i+1)$;
$g(i+2)$;

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
(inputs, zeros) $\mapsto$ (inputs, zeros).

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
$(x, \text{zeros}) \mapsto (p(x), \text{zeros})$.

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

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

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

le and clean:

ishing dirty computation,

outputs back to 0,

ting same operations

outputs in reverse order.

computation:

$\mapsto$

dirt, outputs).

versible computation:

zeros, zeros) $\mapsto$

dirt, outputs).

versible computation:

zeros, zeros) $\mapsto$

zeros, outputs).

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

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

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

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

$\approx$ numb

in origin

This can

than nu

in the or

an:
y computation,
ack to 0,
operations
reverse order.

ion:

uts).

mputation:
s) ↦
uts).

omputation:
s) ↦
puts).

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 $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

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
$\approx$ number of **bit o**
in original p, p^{-1} c

This can be much
than number of **bi**
in the original circ

ation,

as

rder.

n:

n:

16

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 $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

Typically prepare vectors
supported on first 2^n entries
so don't care how permutation
acts on last $2^{n+z} - 2^n$ entries.

17

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.

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 $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

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.

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 $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

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.

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 $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

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.

st circuit for p
circuit for p^{-1},
st reversible circuit for
$) \mapsto (p(x), \text{zeros})$.

reversible bit operations
ffoli gates etc.
ng $\mathbf{C}^{2^{n+z}} \to \mathbf{C}^{2^{n+z}}$.

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

y prepare vectors
ed on first 2^n entries
care how permutation
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 qua

"Hadam
(a_0, a_1)

for p
r p^{-1},
e circuit for
), zeros).

bit operations
etc.
$\to \mathbf{C}^{2^{n+z}}$.

rst 2^n entries is
$) \mapsto$
$\ldots, a_{p^{-1}(2^n-1)})$.

vectors
2^n entries
permutation
$- 2^n$ 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 ope

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a$

or

tions

ies is

$^{n-1)}$).

ion

es.

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, pa

"Hadamard":
$(a_0, a_1) \mapsto (a_0 + a_1, a_0 - a_1$

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

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

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

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

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

Same for qubit 1:
$(a_0, a_1, a_2, a_3) \mapsto$
$(a_0 + a_2, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

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

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

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

g: Number of **qubits**
er of **bit operations**
al p, p^{-1} circuits.

n be much larger
mber of **bits stored**
riginal circuits.

seful techniques
ress into fewer qubits,
n these lose time.
ubtle tradeoffs.

poly-time" analyses
re about this,
ous cryptanalysis
more precise.

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
$(1, 0, 0, .$

Measuri
always p

Measuri
can proc
Pr[outpu

of **qubits**
operations
circuits.

larger
its stored
uits.

niques
fewer qubits,
se time.
offs.

analyses
his,
nalysis
cise.

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

Measuring $(1, 0, 0,$
always produces 0

Measuring $(1, 1, 1,$
can produce any o
$\Pr[\text{output} = q] =$

s

s

ts,

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

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

Measuring $(1, 1, 1, \ldots, 1)$
can produce any output:
$\Pr[\text{output} = q] = 1/2^n$.

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

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

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

antum operations, part 2

ard":

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

$a_2, a_3) \mapsto$

$, a_0 - a_1, a_2 + a_3, a_2 - a_3)$.

r qubit 1:

$a_2, a_3) \mapsto$

$, a_1 + a_3, a_0 - a_2, a_1 - a_3)$.

and then qubit 1:

$a_2, a_3) \mapsto$

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

$+ a_2 + a_3, a_0 - a_1 + a_2 - a_3,$

$- a_2 - a_3, a_0 - a_1 - a_2 + a_3)$.

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

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

Simon's

Assume
satisfies
for every
Can we
given a

erations, part 2

$a_1, a_0 - a_1)$.

$a_2 + a_3, a_2 - a_3)$.

$a_0 - a_2, a_1 - a_3)$.

qubit 1:

$_2 + a_3, a_2 - a_3) \mapsto$

$a_0 - a_1 + a_2 - a_3,$

$a_0 - a_1 - a_2 + a_3)$.

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

Aside from "normalization"
(irrelevant to measurement),
have Hadamard $=$ 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
satisfies $f(x) = f($
for every $x \in \{0, 1$
Can we find this p
given a fast circuit

art 2

).

$- a_3$).

$a_3) \mapsto$

$a_2 - a_3$,

$a_2 + a_3$).

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

Aside from "normalization"
(irrelevant to measurement),
have Hadamard $=$ 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?

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 $=$ 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?

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 $=$ 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?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

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 $=$ 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?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

n times: e.g.,

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

ng $(1, 0, 0, \ldots, 0)$

roduces 0.

ng $(1, 1, 1, \ldots, 1)$

duce any output:

$\text{ut} = q] = 1/2^n$.

om "normalization"

nt to measurement),

$\text{damard} = \text{Hadamard}^{-1}$,

work backwards

niform superposition"

$\ldots, 1)$ to "pure state"

$\ldots, 0)$.

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?

We don't have enough data
if f has many periods.
Assume: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's

far fewe

if n is la

reversibi

.g.,
$(1,1,1,\ldots,1)$.

$\ldots,0)$
.

$\ldots,1)$
output:
$1/2^n$.

alization"
surement),
Hadamard^{-1},
kwards
erposition"
"pure state"

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: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm
far fewer qubit op
if n is large and
reversibility overhe

, 1).

,

$^{-1}$,

"

e"

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: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

Say f maps n bits to m bits using
z "ancilla" bits for reversibility.

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

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: $\{\text{periods}\} = \{0, s\}$.

Traditional solution:
Compute f for many inputs,
sort, analyze collisions.
Success probability is very low
until #inputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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.

algorithm

nonzero $s \in \{0,1\}^n$
$f(x) = f(x \oplus s)$
$x \in \{0,1\}^n$.
find this period s,
fast circuit for f?

t have enough data
many periods.
$\{\text{periods}\} = \{0, s\}$.

nal solution:
e f for many inputs,
alyze collisions.
probability is very low
nputs approaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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 fa
for rever
1 in posi
moves to

Note sym
1 at $(q,$
1 at $(q \oplus$

$s \in \{0,1\}^n$
$x \oplus s)$
$\}^n$.
eriod s,
t for f?

ough data
ods.
$\} = \{0, s\}$.

n:
ny inputs,
ions.
y is very low
roaches $2^{n/2}$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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
for reversible f co
1 in position $(q, 0,$
moves to position

Note symmetry be
1 at $(q, f(q), 0)$ a
1 at $(q \oplus s, f(q),$

21

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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.

22

Apply fast vector permutatio
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 $(q \oplus s, f(q), 0)$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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 $(q \oplus s, f(q), 0)$.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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.

Simon's algorithm uses
far fewer qubit operations
if n is large and
reversibility overhead is low.

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.

algorithm uses
r qubit operations
rge and
lity overhead is low.

aps n bits to m bits using
a" bits for reversibility.

$n + m + z$ qubits
zero state:
$1, 0, 0, \ldots)$.

old Hadamard
first n qubits
form superposition:
$\ldots, 1, 0, 0, \ldots)$
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.

Example

$f(0) = 4$
$f(1) = 7$
$f(2) = 2$
$f(3) = 3$
$f(4) = 7$
$f(5) = 4$
$f(6) = 3$
$f(7) = 2$

uses
erations

ead is low.

to m bits using
r reversibility.

z qubits

.

ard
bits
rposition:
$\ldots$)
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.

Example, 3 bits to

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.

using

ity.

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.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.

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.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.

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 $(q \oplus s, f(q), 0)$.

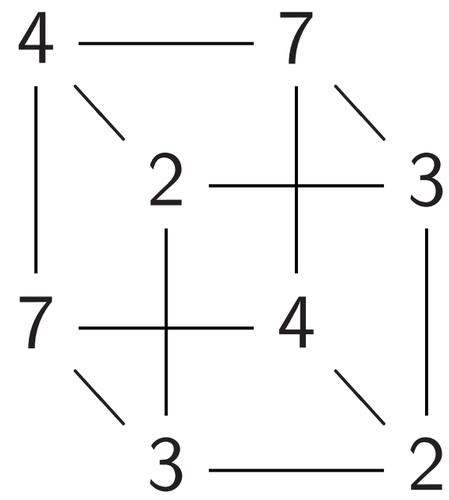
Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



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 $(q \oplus s, f(q), 0)$.

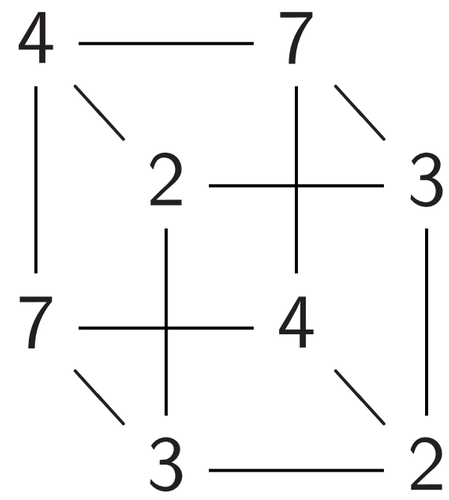
Apply n-fold Hadamard.

Measure. By symmetry,
output is orthogonal to s.

Repeat $n + 10$ times.
Use Gaussian elimination
to (probably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

st vector permutation

rsible f computation:

ition $(q, 0, 0)$

o position $(q, f(q), 0)$.

mmetry between

$f(q), 0)$ and

$\oplus\, s, f(q), 0)$.

-fold Hadamard.

e. By symmetry,

s orthogonal to s.

$n + 10$ times.

ssian elimination

ably) find s.

Example, 3 bits to 3 bits:

$f(0) = 4.$

$f(1) = 7.$

$f(2) = 2.$

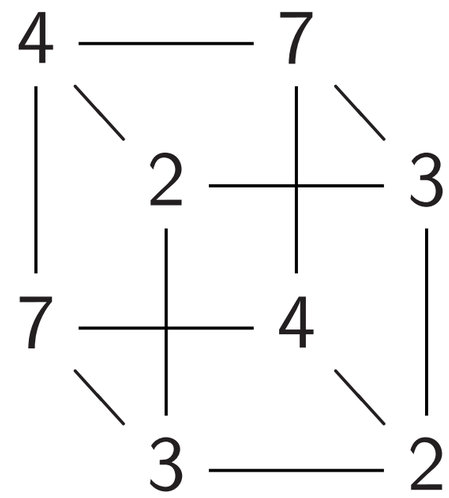
$f(3) = 3.$

$f(4) = 7.$

$f(5) = 4.$

$f(6) = 3.$

$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

Step 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

permutation

mputation:

, 0)

$(q, f(q), 0)$.

etween

nd

0).

mard.

metry,

nal to s.

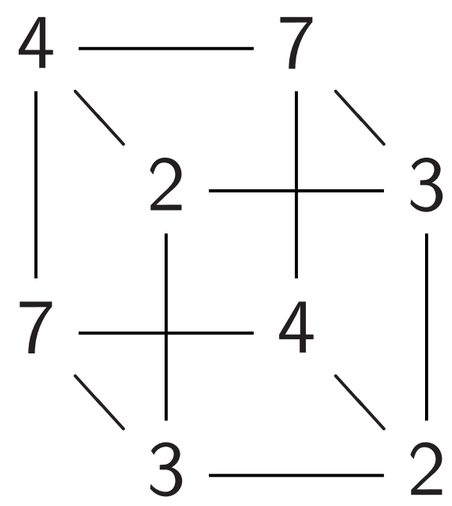
nes.

ination

s.

Example, 3 bits to 3 bits:

$f(0) = 4$.
$f(1) = 7$.
$f(2) = 2$.
$f(3) = 3$.
$f(4) = 7$.
$f(5) = 4$.
$f(6) = 3$.
$f(7) = 2$.



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

Step 1. Set up pu

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.

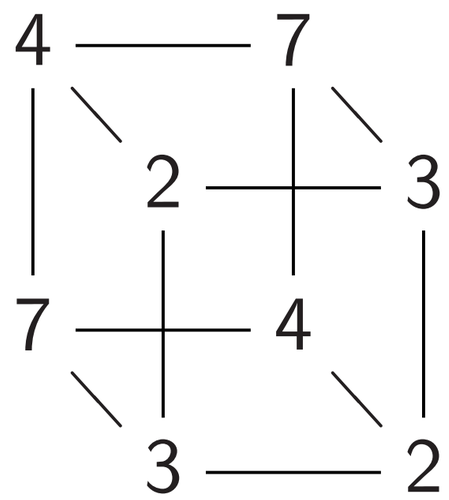
on

:

0)).

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

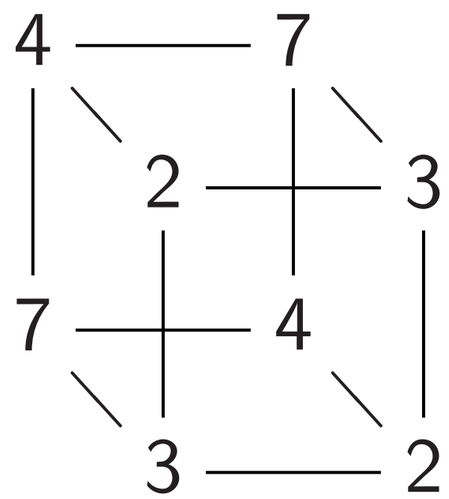
Let's watch Simon's algorithm for f, using 6 qubits.

Step 1. Set up pure zero sta

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.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

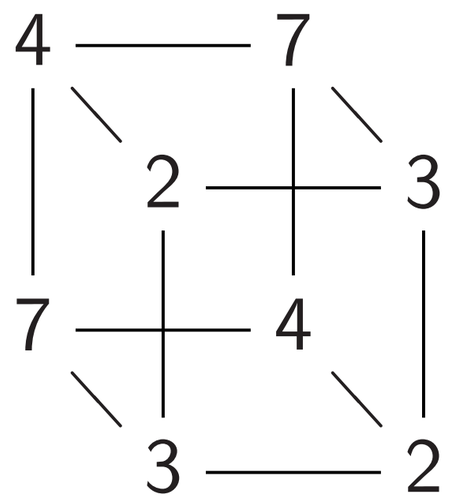
Let's watch Simon's algorithm
for f, using 6 qubits.

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.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

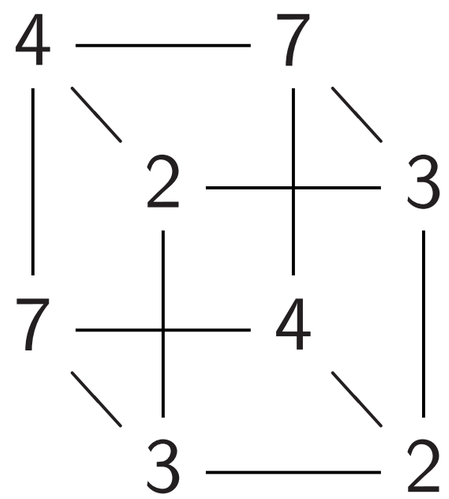
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 2. Hadamard on qubit 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.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

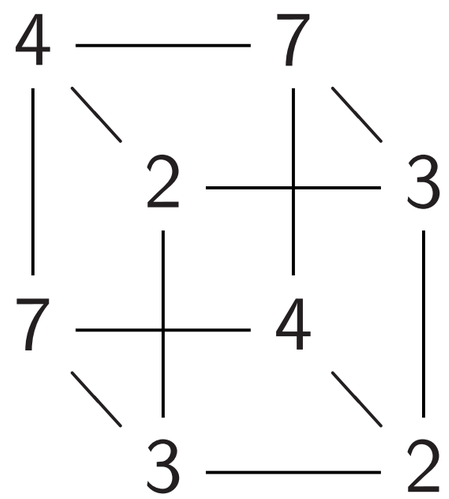
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 3. Hadamard on qubit 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.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

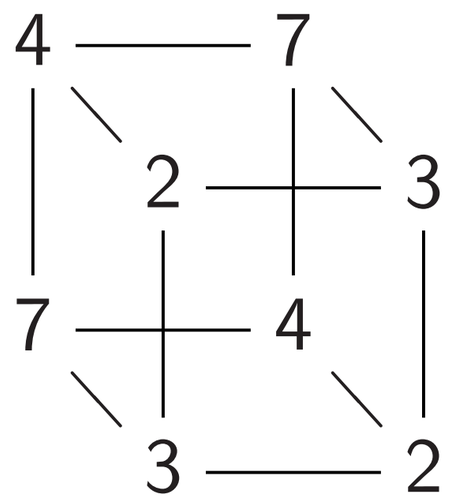
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 4. Hadamard on qubit 2:

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.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

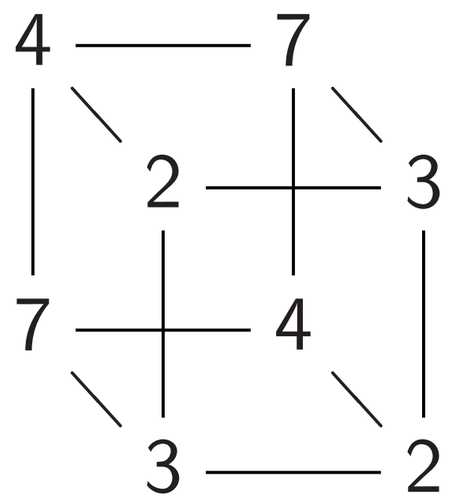
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 5. $(q, 0) \mapsto (q, f(q))$:

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, 1, 0, 0, 1, 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, 1, 0, 0, 1, 0, 0, 0.

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

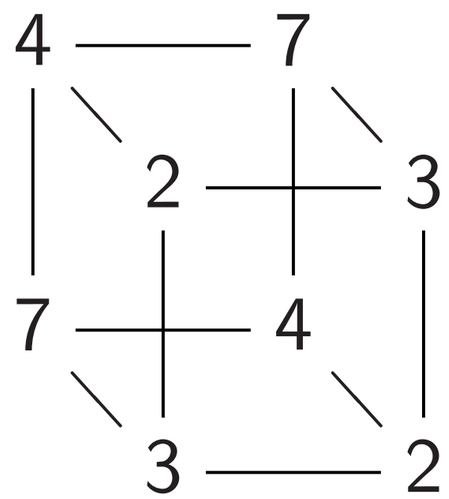
Step 6. Hadamard on qubit 0:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 1, 1, 0, 0, 1, \overline{1},$
$0, 0, 1, \overline{1}, 0, 0, 1, 1,$
$1, 1, 0, 0, 1, \overline{1}, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$1, \overline{1}, 0, 0, 1, 1, 0, 0.$

Notation: $\overline{1} = -1.$

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

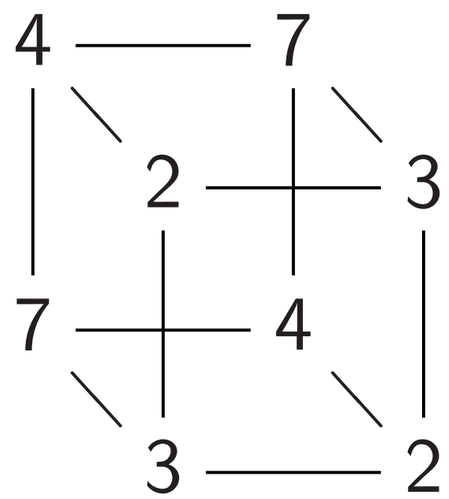
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 7. Hadamard on qubit 1:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$1, 1, \overline{1}, \overline{1}, 1, \overline{1}, \overline{1}, 1,$
$1, \overline{1}, \overline{1}, 1, 1, 1, \overline{1}, \overline{1},$
$1, 1, 1, 1, 1, \overline{1}, 1, \overline{1},$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$1, \overline{1}, 1, \overline{1}, 1, 1, 1, 1.$

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

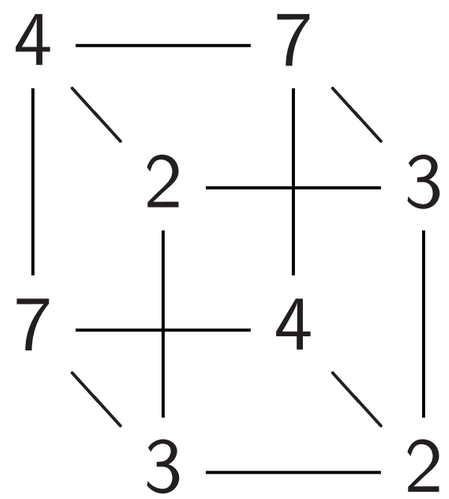
Let's watch Simon's algorithm
for f, using 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$2, 0, 2, 0, 0, 2, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Example, 3 bits to 3 bits:

$f(0) = 4.$
$f(1) = 7.$
$f(2) = 2.$
$f(3) = 3.$
$f(4) = 7.$
$f(5) = 4.$
$f(6) = 3.$
$f(7) = 2.$



Complete table shows that
$f(x) = f(x \oplus 5)$ for all x.

Let's watch Simon's algorithm
for f, using 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$2, 0, 2, 0, 0, 2, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

, 3 bits to 3 bits:

4.

7.

2.

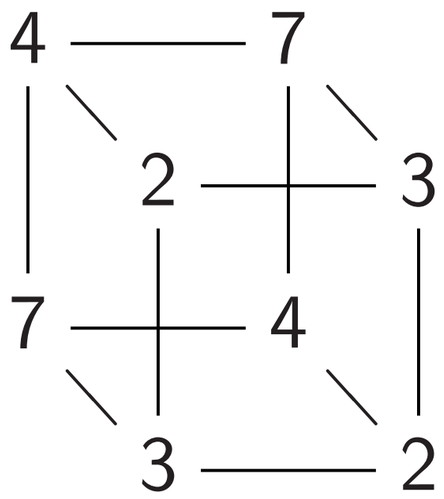
3.

7.

4.

3.

2.



e table shows that
$f(x \oplus 5)$ for all x.

tch Simon's algorithm
sing 6 qubits.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

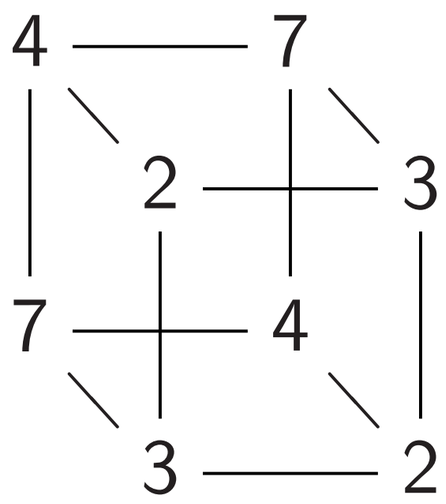
Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's

Assume:
has $f(s)$

Traditio
compute
hope to
Success
until #i

3 bits:



ows that
or all x.

n's algorithm
its.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$2, 0, 2, 0, 0, 2, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

Assume: unique s
has $f(s) = 0$.

Traditional algorit
compute f for ma
hope to find outpu
Success probabilit
until #inputs appr

3

2

m

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$

$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$

$2, 0, 2, 0, 0, 2, 0, 2,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$0, 0, 0, 0, 0, 0, 0, 0,$

$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.

First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

Grover's algorithm

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

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

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$2, 0, 2, 0, 0, 2, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.

First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

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.

Step 8. Hadamard on qubit 2:

$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, \overline{2},$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$2, 0, 2, 0, 0, 2, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2}.$

Step 9. Measure.
First 3 qubits are uniform random vector orthogonal to 101: i.e., 000, 010, 101, or 111.

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.

Hadamard on qubit 2:

$, 0, 0, 0, 0,$
$, 0, 0, 0, 0,$
$, 0, \overline{2}, 0, \overline{2},$
$, 0, \overline{2}, 0, 2,$
$, 0, 2, 0, 2,$
$, 0, 0, 0, 0,$
$, 0, 0, 0, 0,$
$, 0, \overline{2}, 0, \overline{2}.$
$, 0, 2, 0, 2.$

Measure.

qubits are uniform random

rthogonal to 101: i.e.,

0, 101, or 111.

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 fro
over all

Step 1:

$b_q = -a$

$b_q = a_q$

This is f

Step 2:

Negate

This is a

Repeat S
about 0.

Measure
With hig

d on qubit 2:

uniform random
to 101: i.e.,
111.

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
over all n-bit strin

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

Step 2: "Grover d
Negate a around i
This is also fast.

Repeat Step 1 + S
about $0.58 \cdot 2^{0.5n}$

Measure the n qub
With high probabi

2:

ndom

e.,

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 superpos
over all n-bit strings q.

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 $0.58 \cdot 2^{0.5n}$ times.

Measure the n qubits.
With high probability this fi

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
$b_q = -a_q$ 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 $0.58 \cdot 2^{0.5n}$ times.

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

algorithm

unique $s \in \{0,1\}^n$
$= 0$.

nal algorithm to find s:
e f for many inputs,
find output 0.
probability is very low
nputs approaches 2^n.

algorithm takes only $2^{n/2}$
e computations of f.
y: reversibility overhead
enough that this
eats traditional algorithm.

Start from uniform superposition over all n-bit strings q.

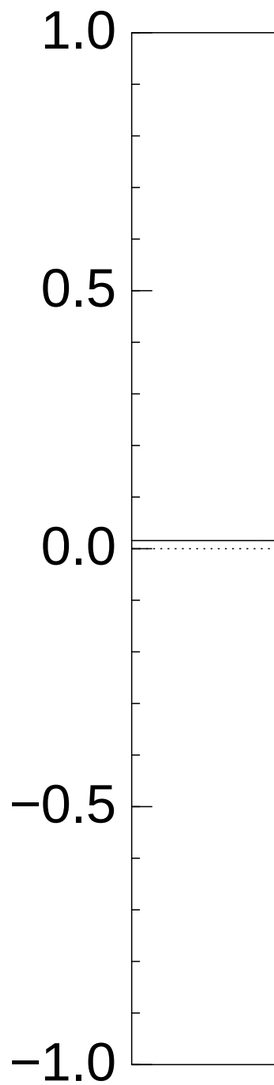
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ times.

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

Normaliz
for an ex
after 0 s



n

$\in \{0,1\}^n$

hm to find s:
ny inputs,
ut 0.
y is very low
roaches 2^n.

takes only $2^{n/2}$
ations of f.
ility overhead
at this
ional algorithm.

Start from uniform superposition over all n-bit strings q.

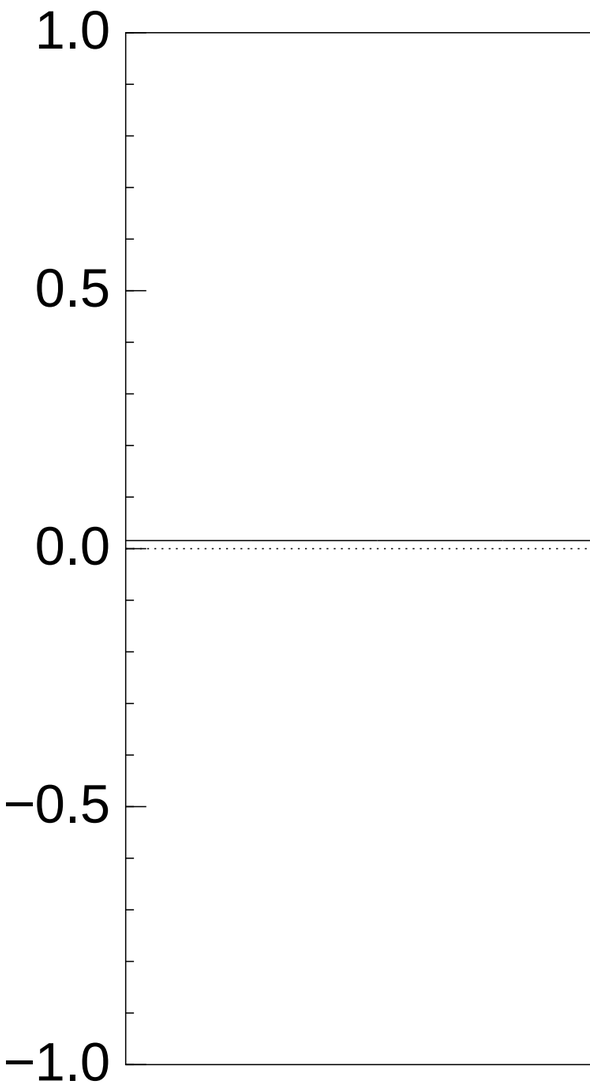
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ times.

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

Normalized graph
for an example wit
after 0 steps:



Start from uniform superposition over all n-bit strings q.

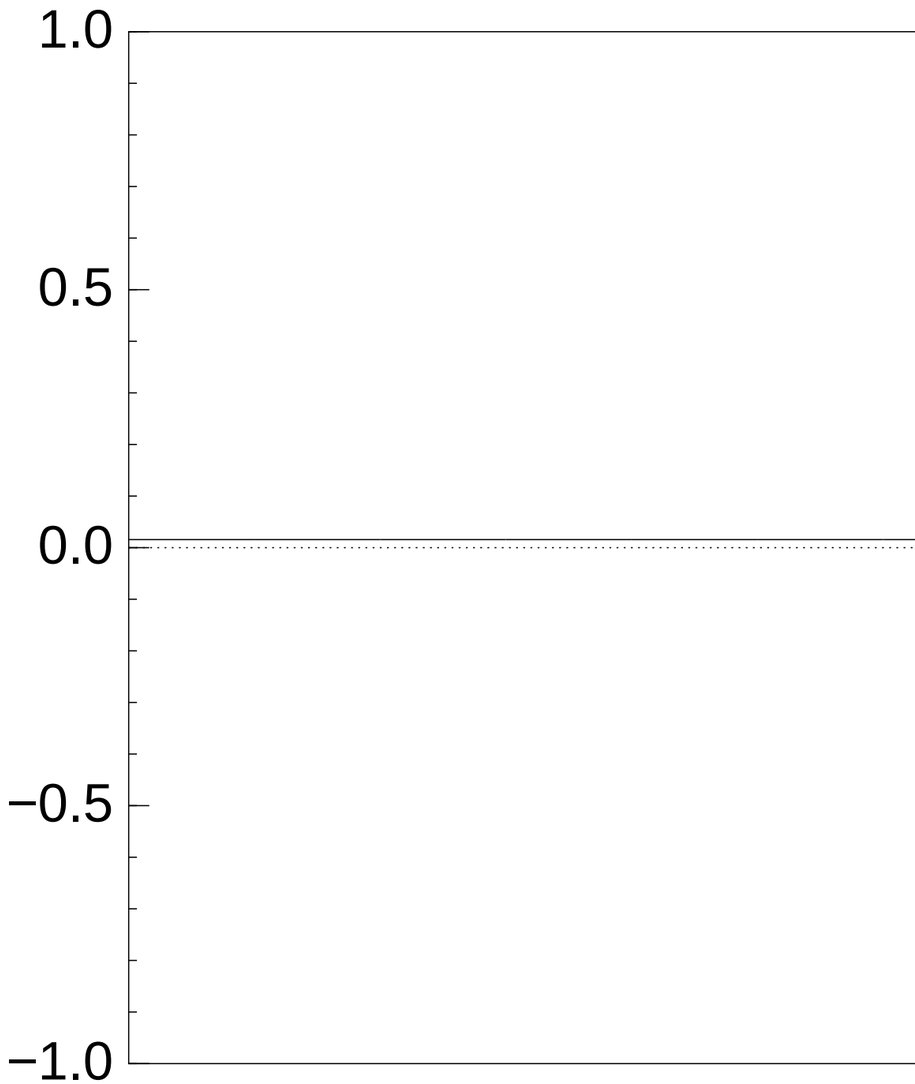
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 over all n-bit strings q.

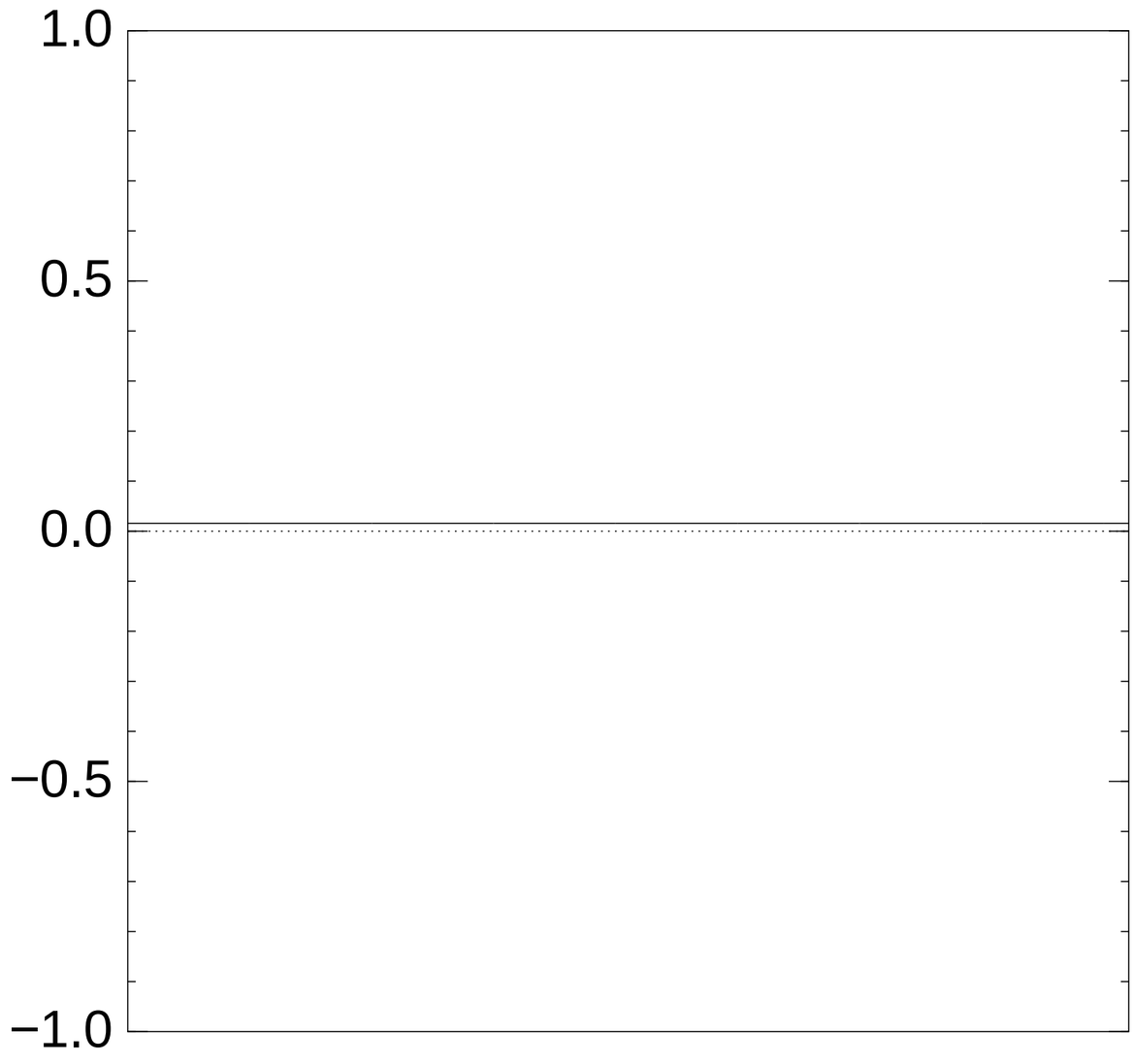
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 over all n-bit strings q.

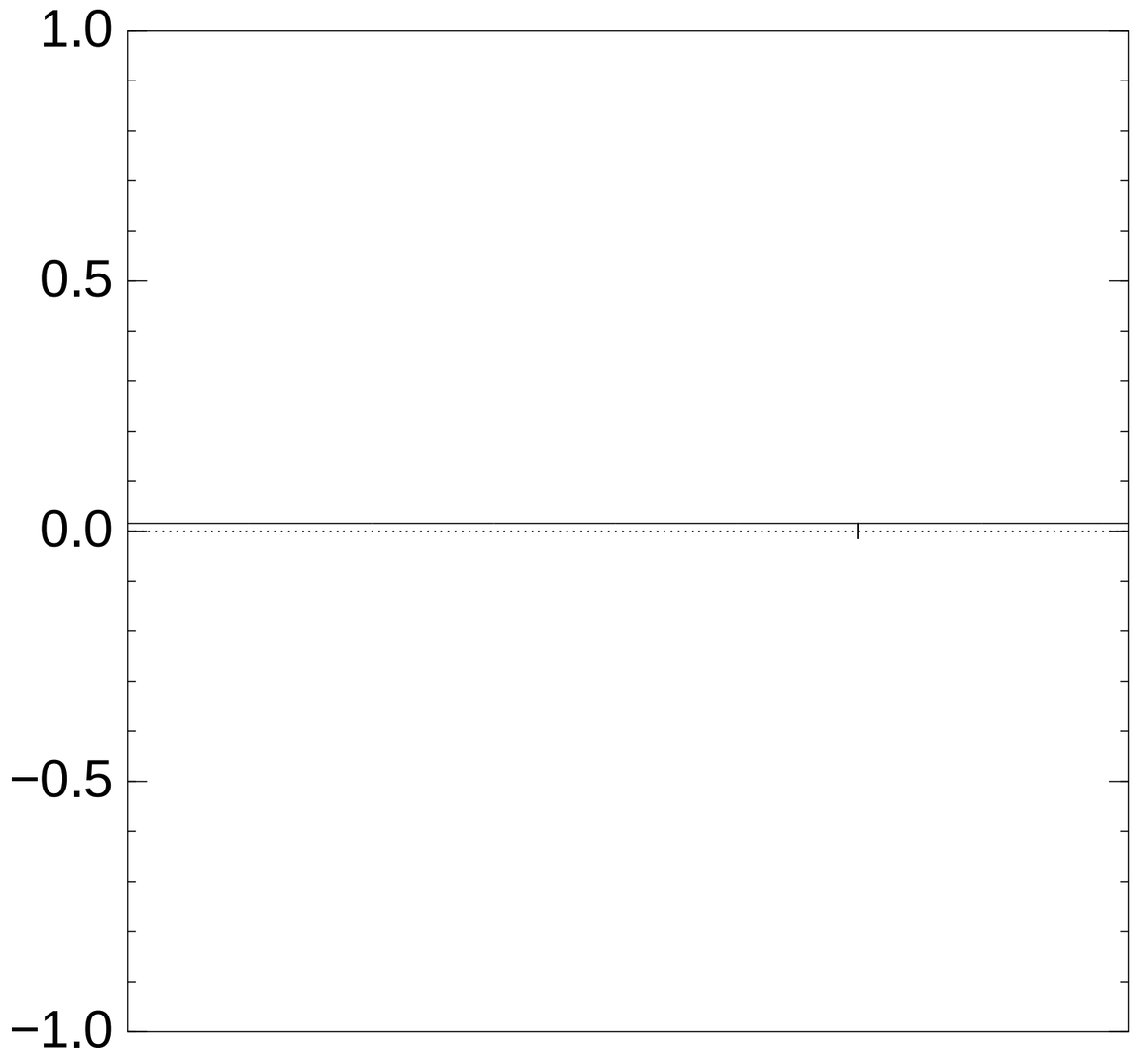
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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:



Start from uniform superposition over all n-bit strings q.

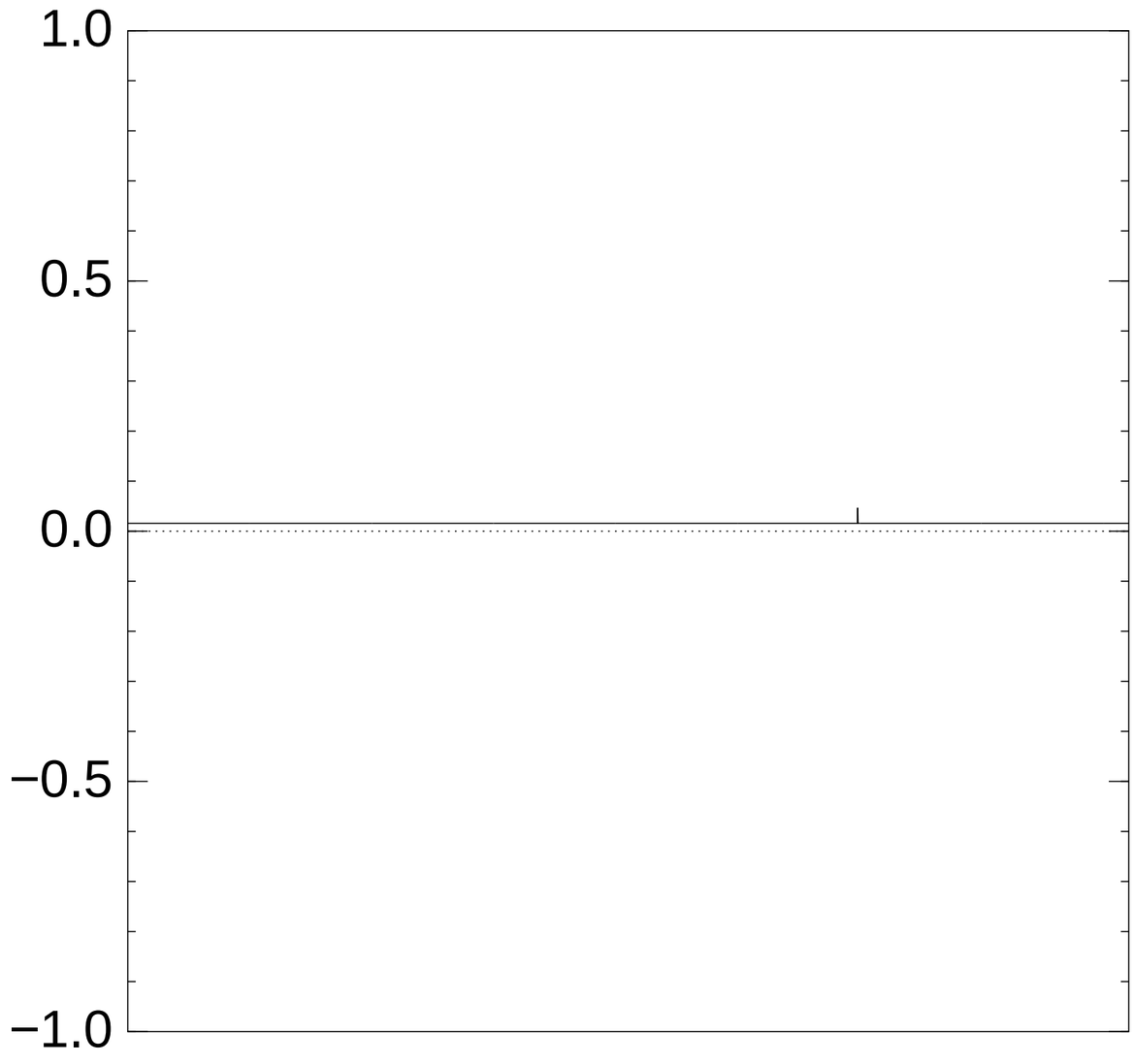
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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
over all n-bit strings q.

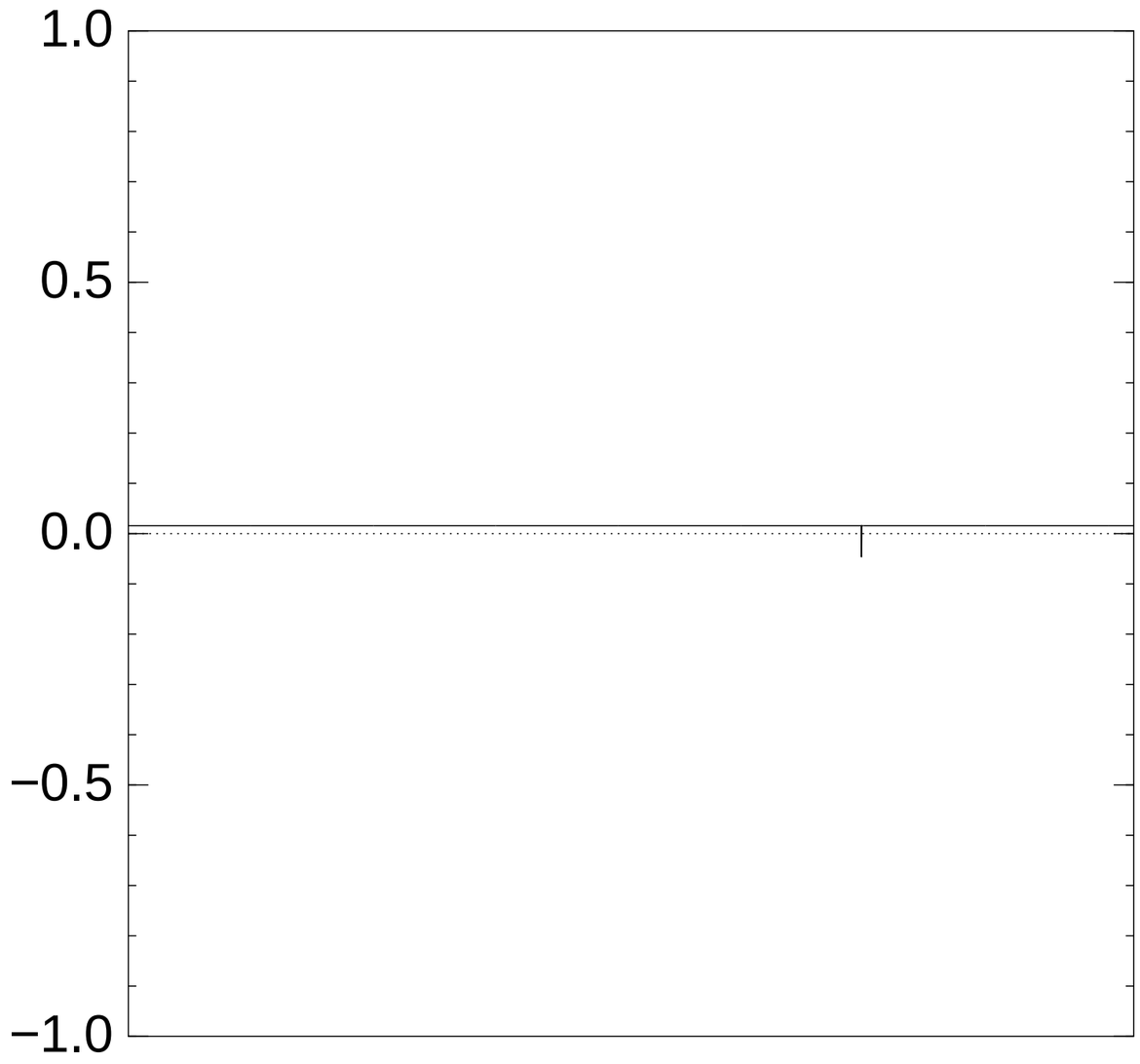
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 over all n-bit strings q.

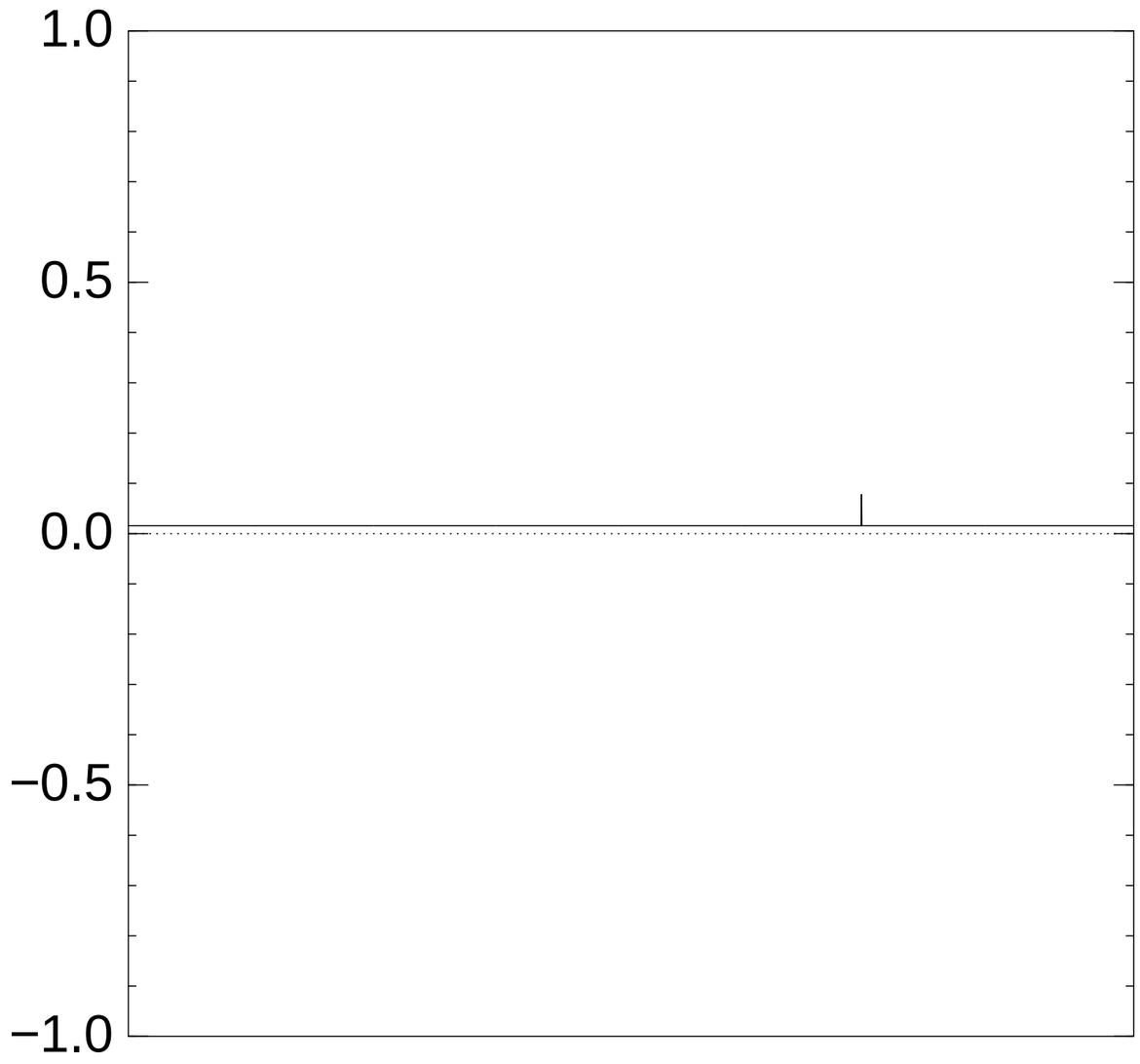
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $2 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

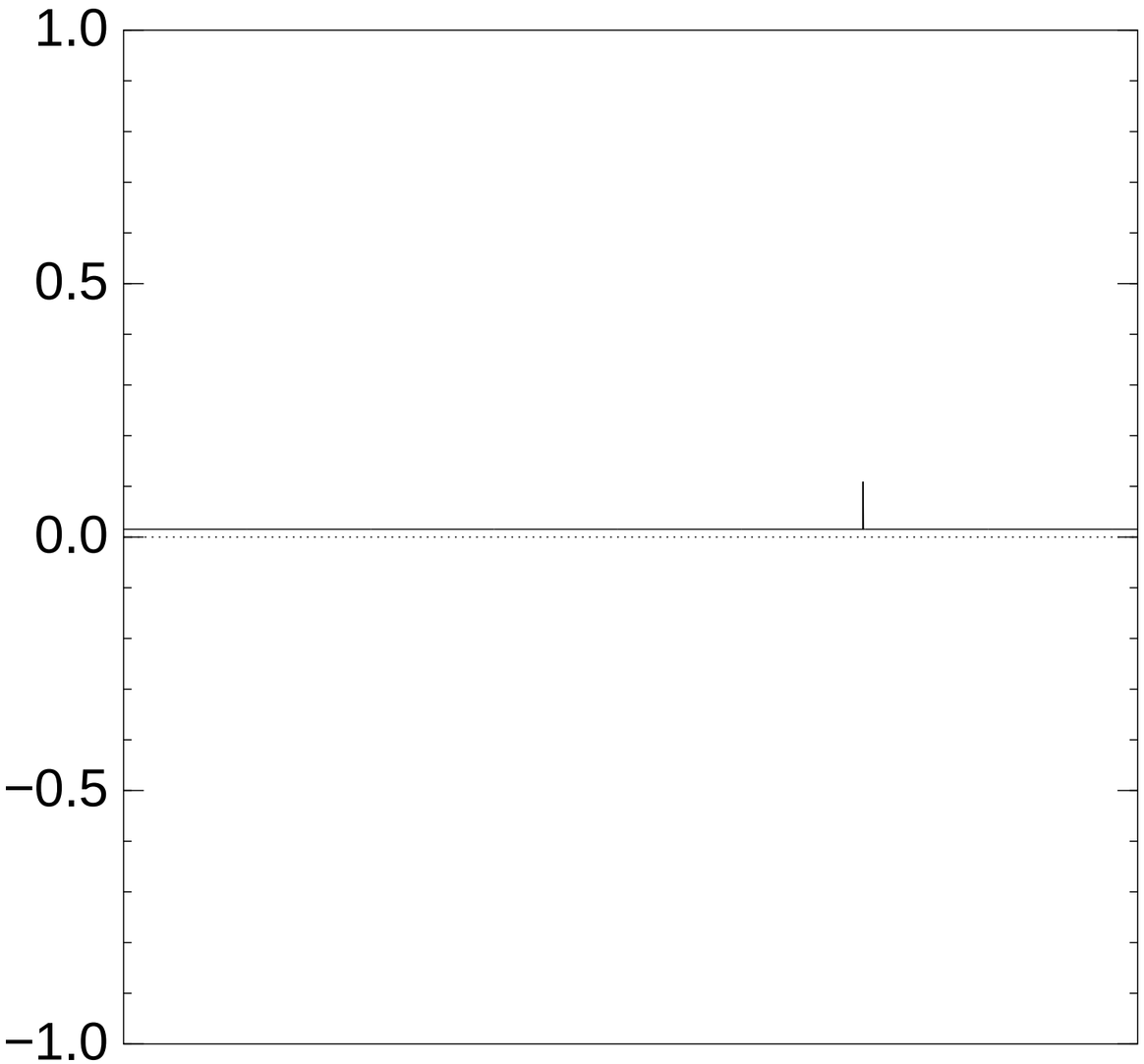
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $3 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

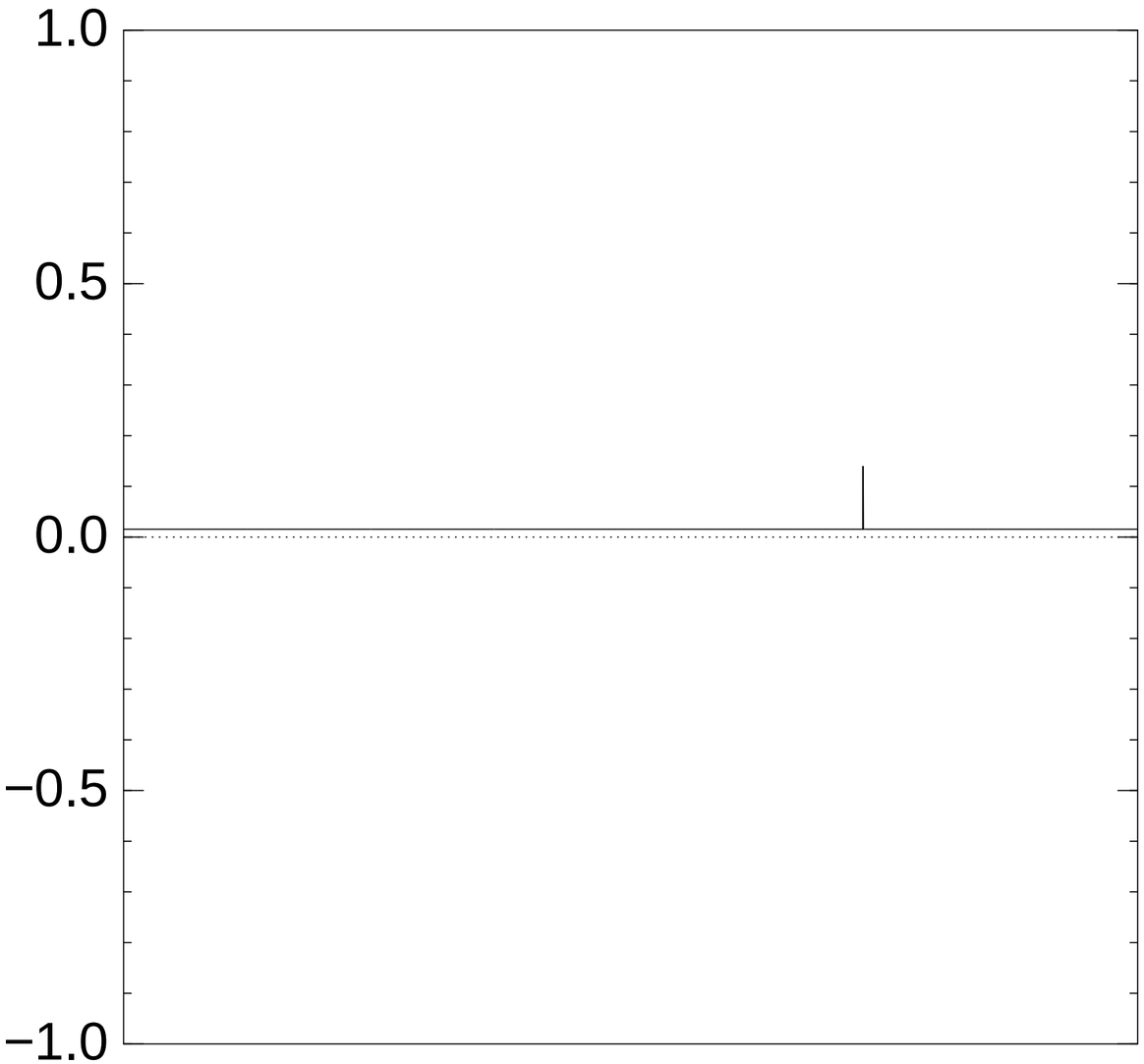
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $4 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

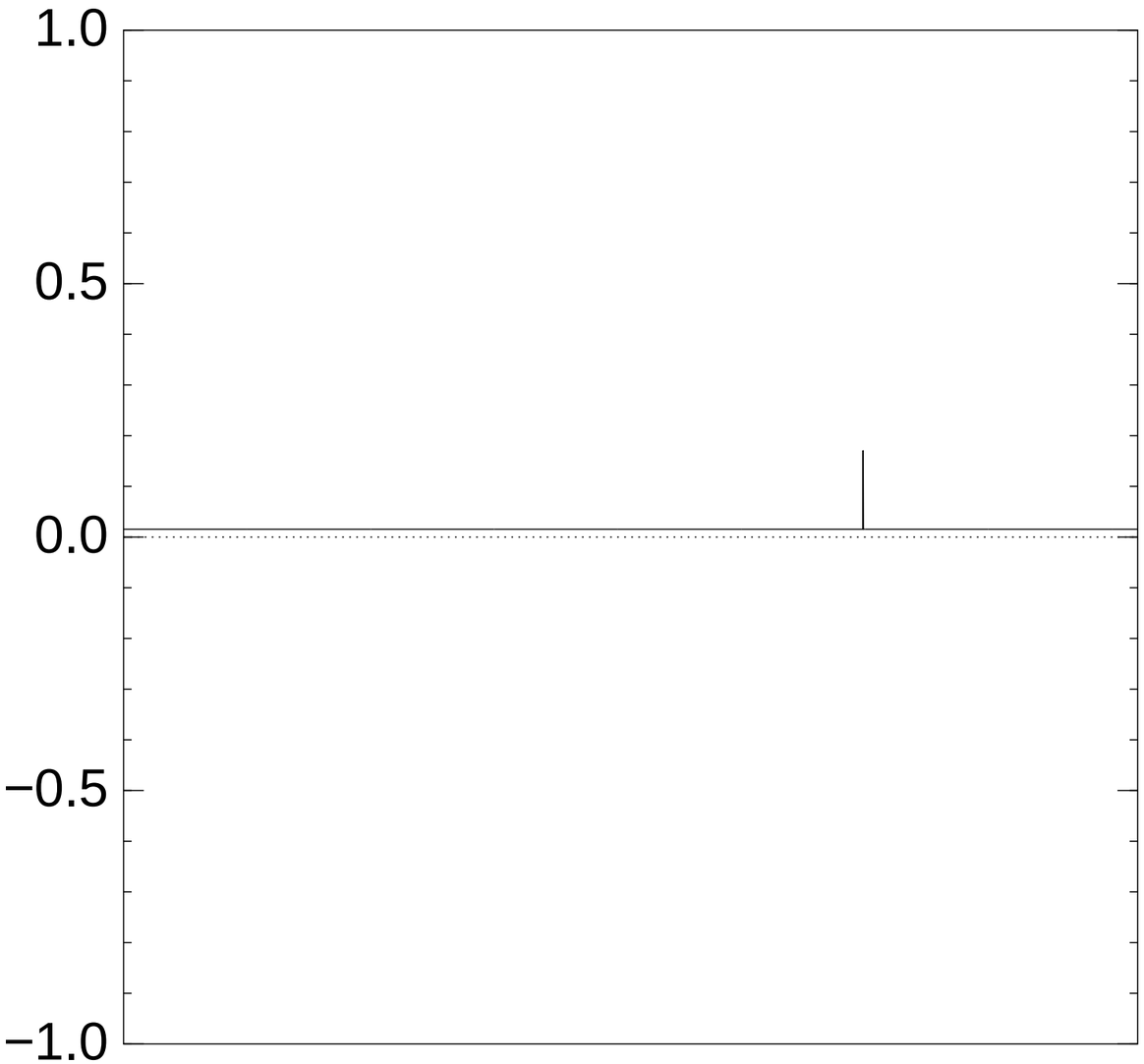
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $5 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

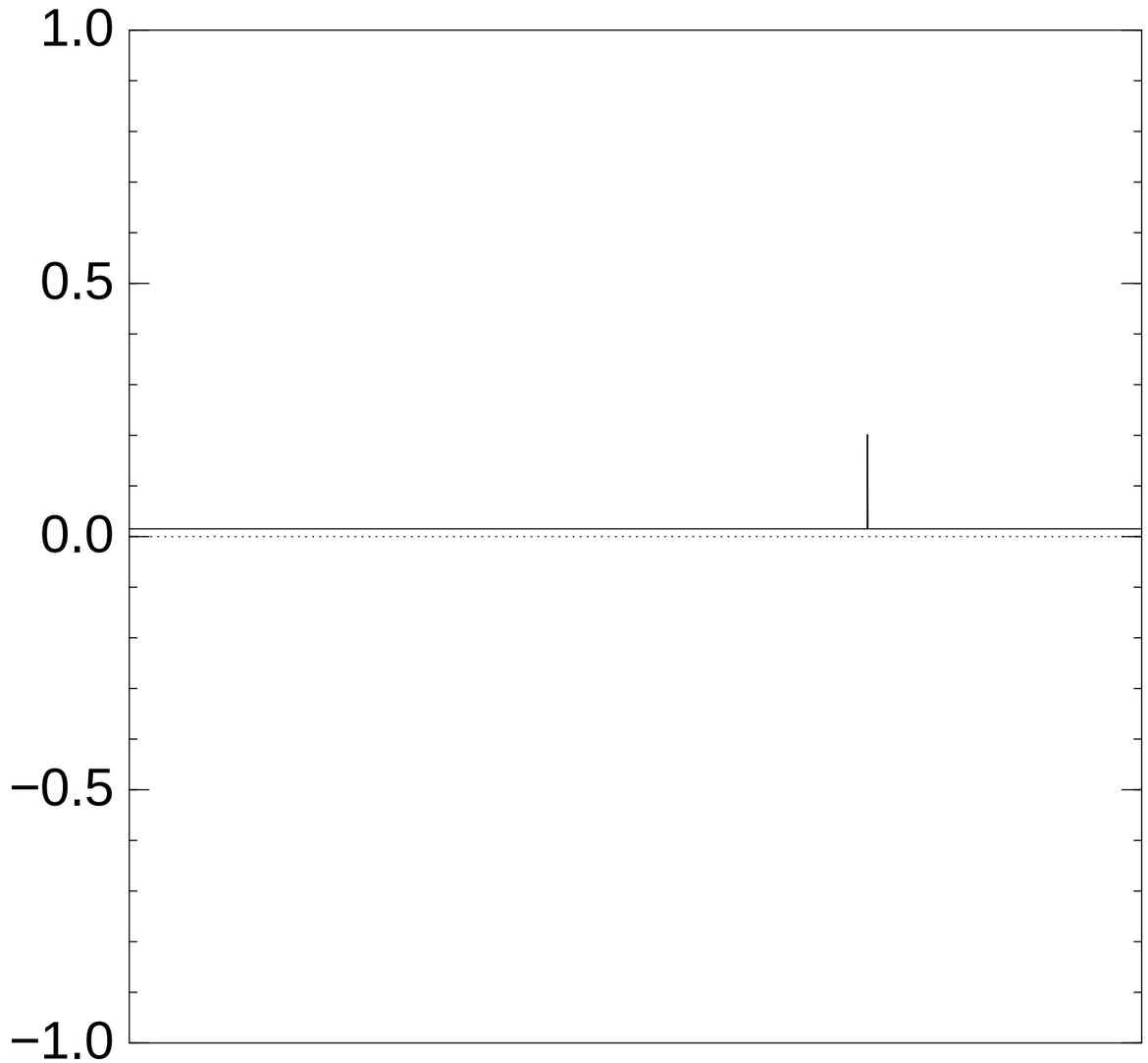
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $6 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

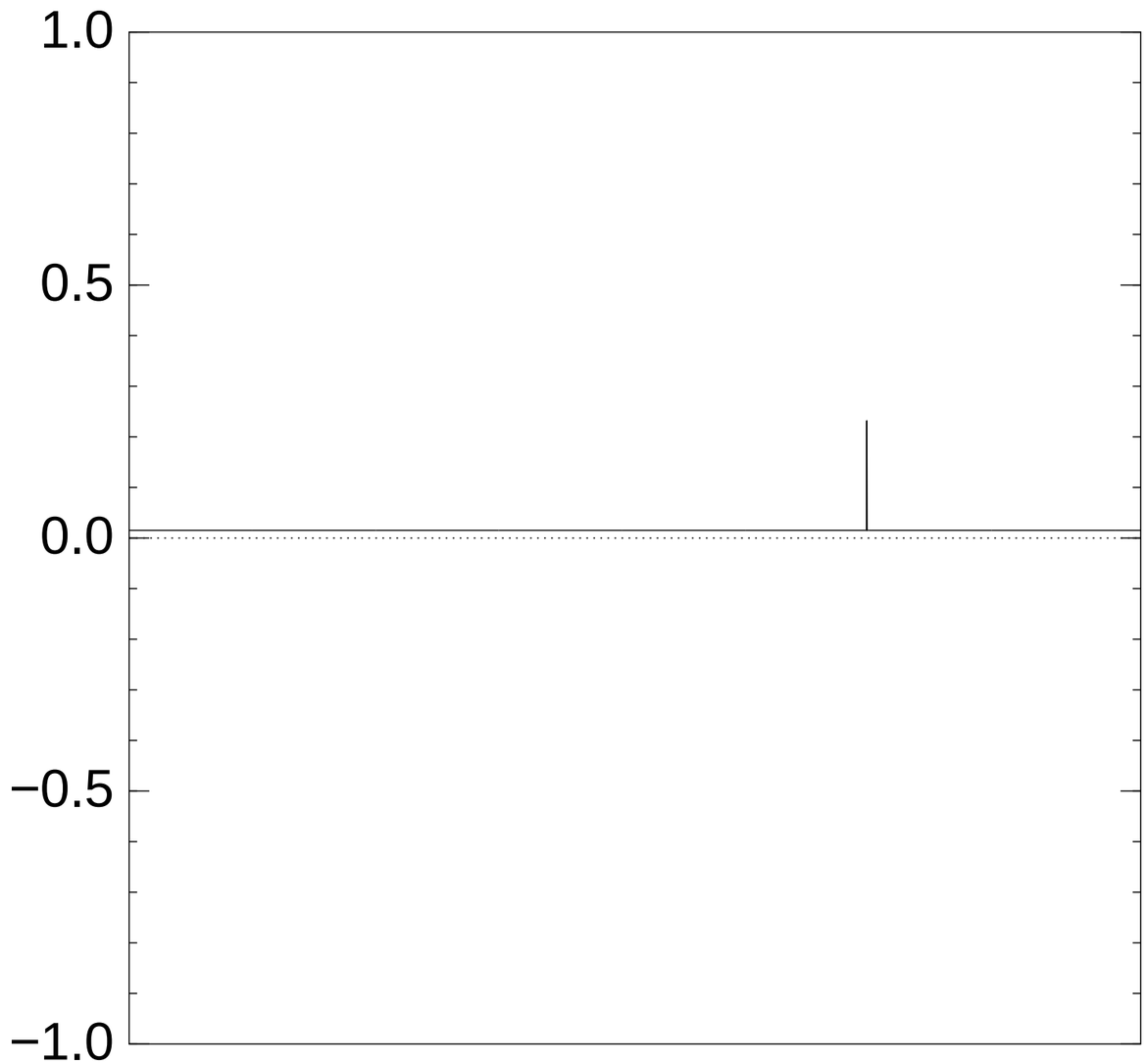
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $7 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

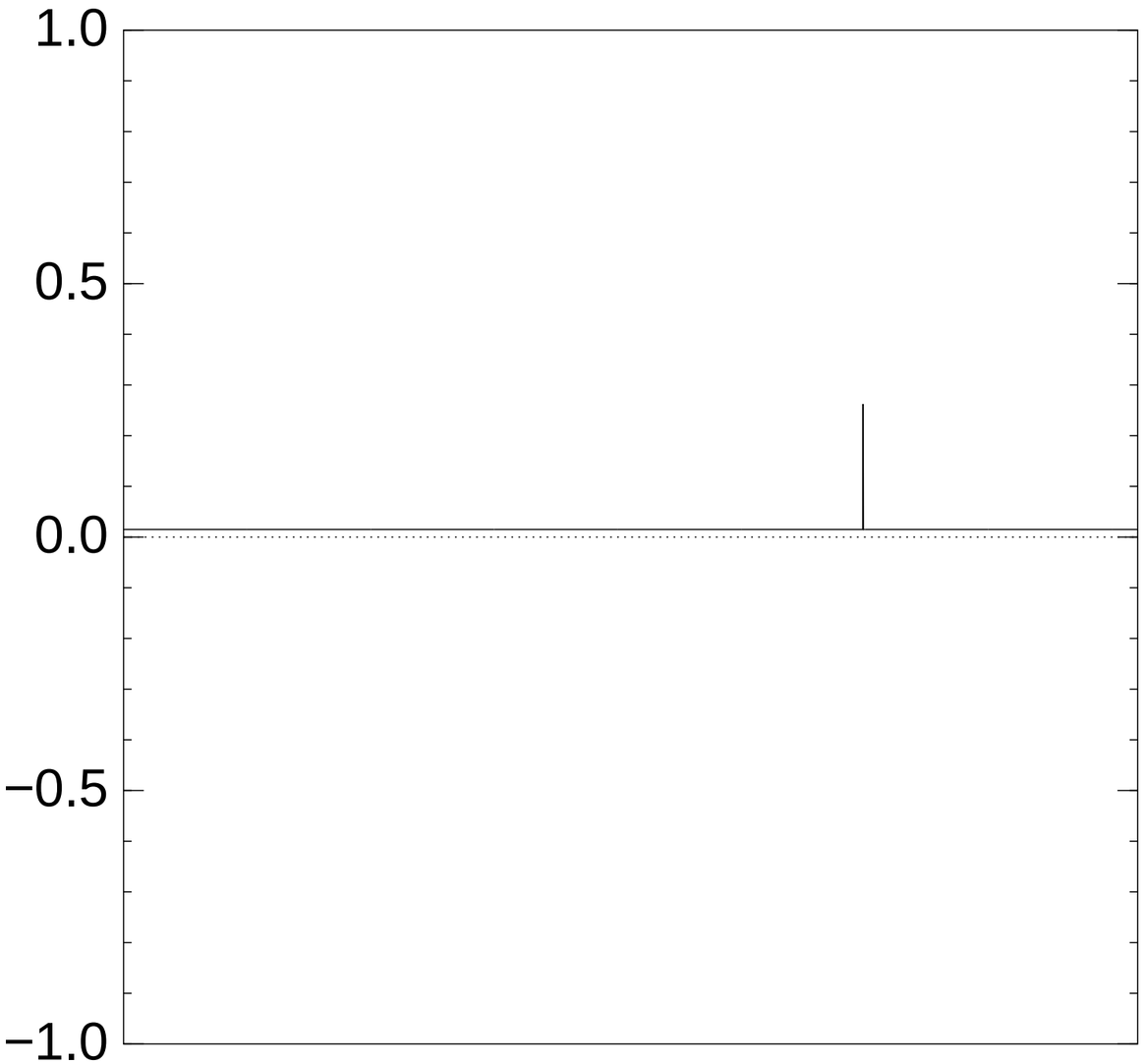
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $8 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

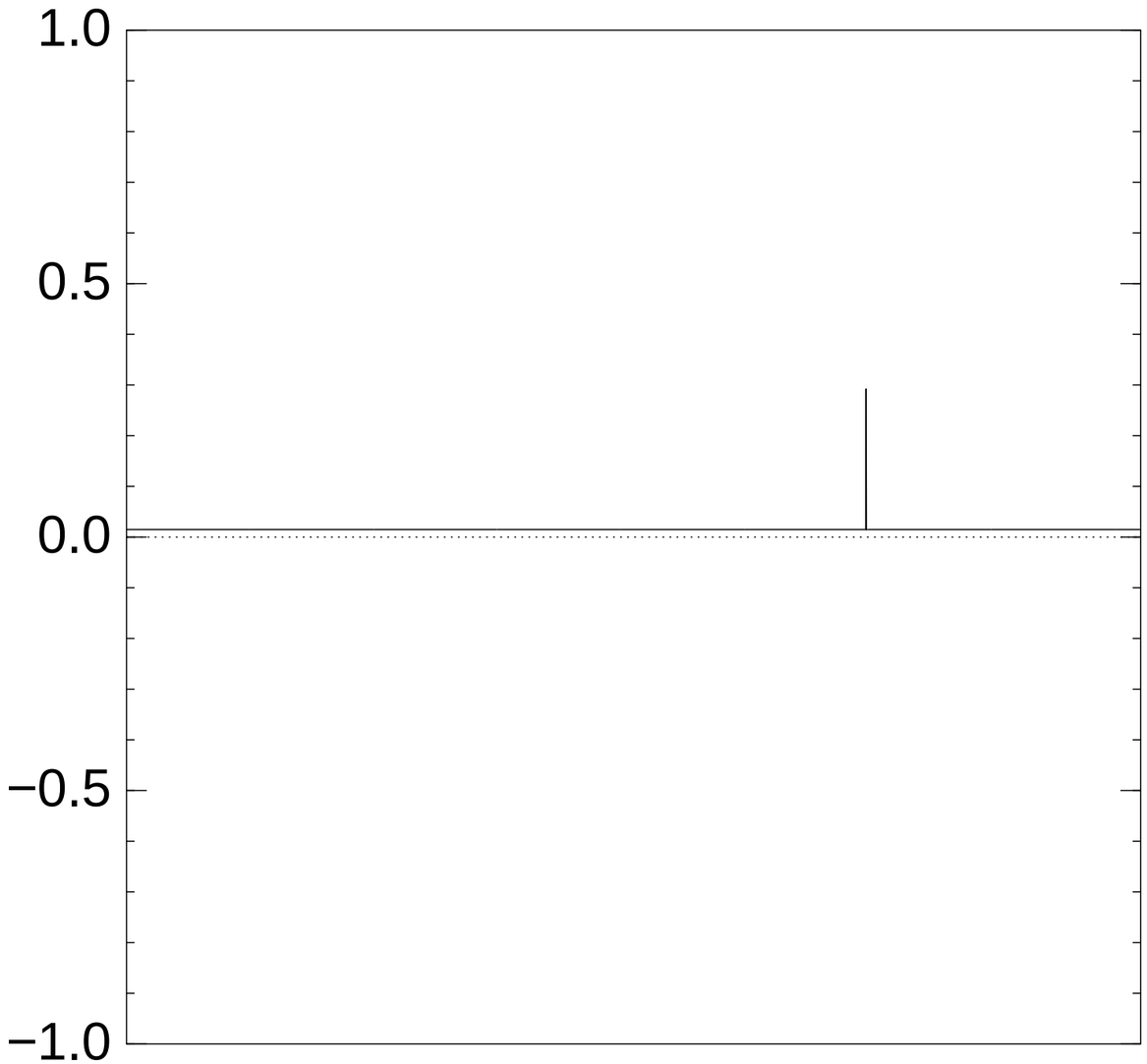
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $9 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

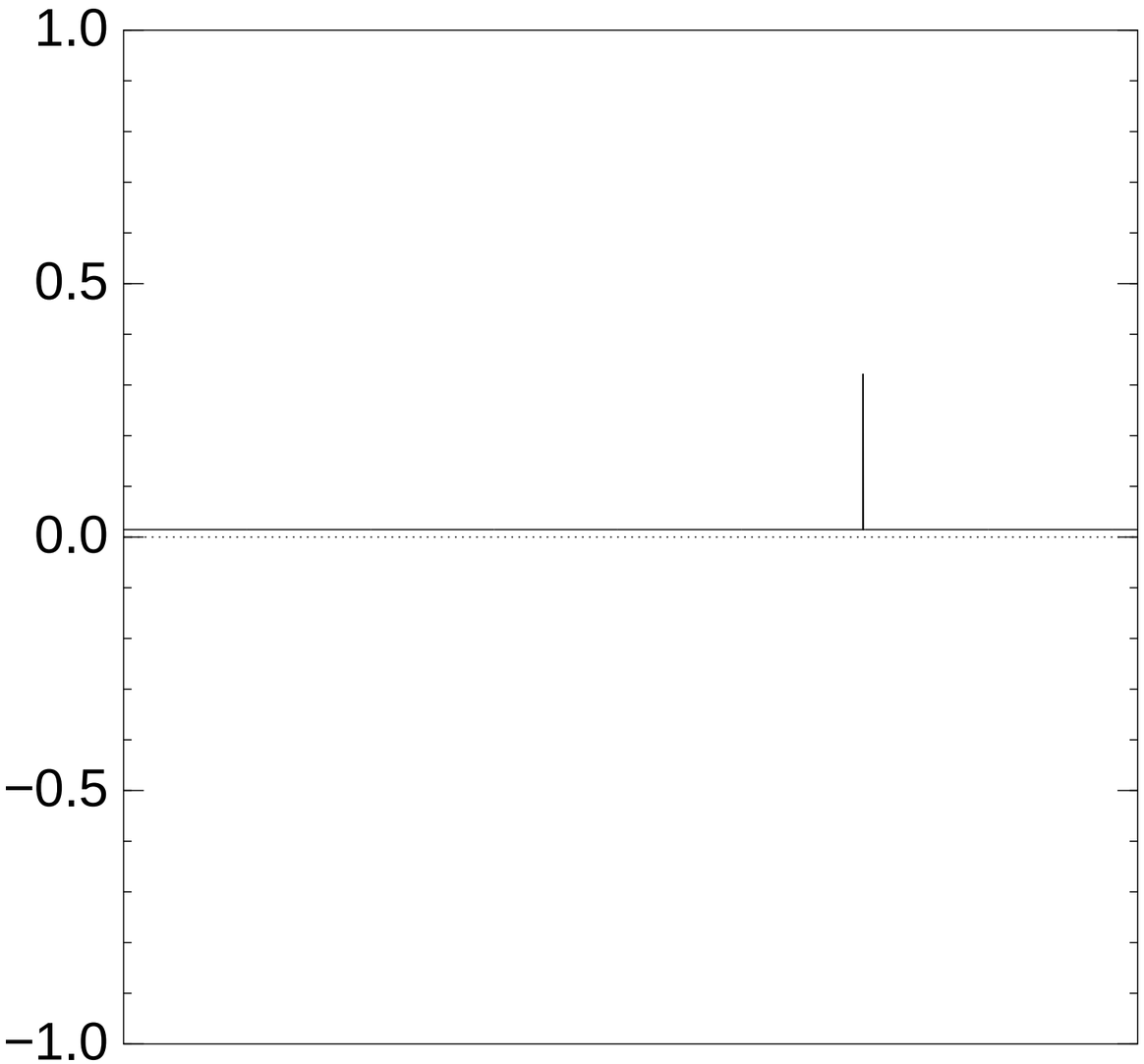
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $10 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

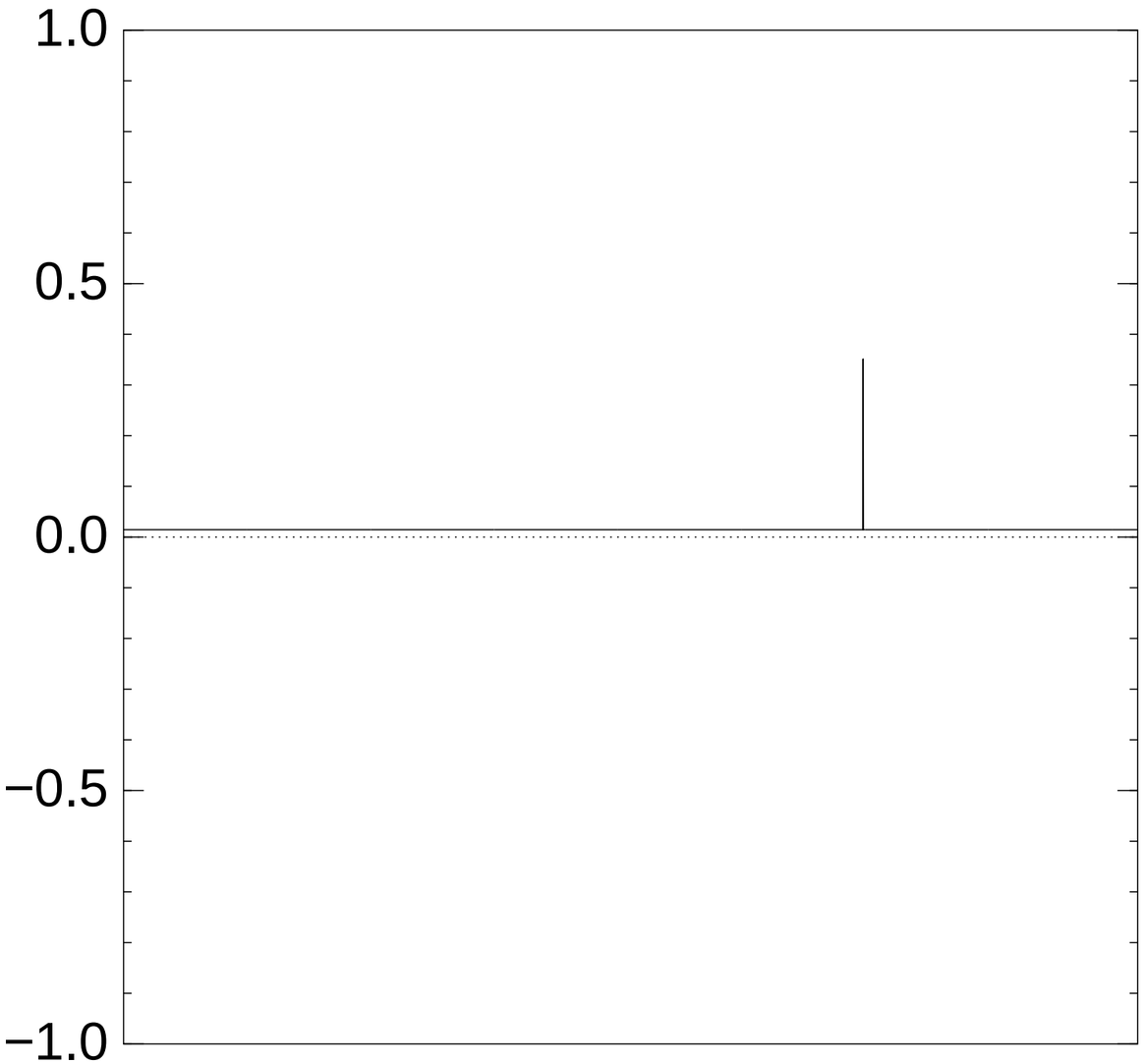
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $11 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

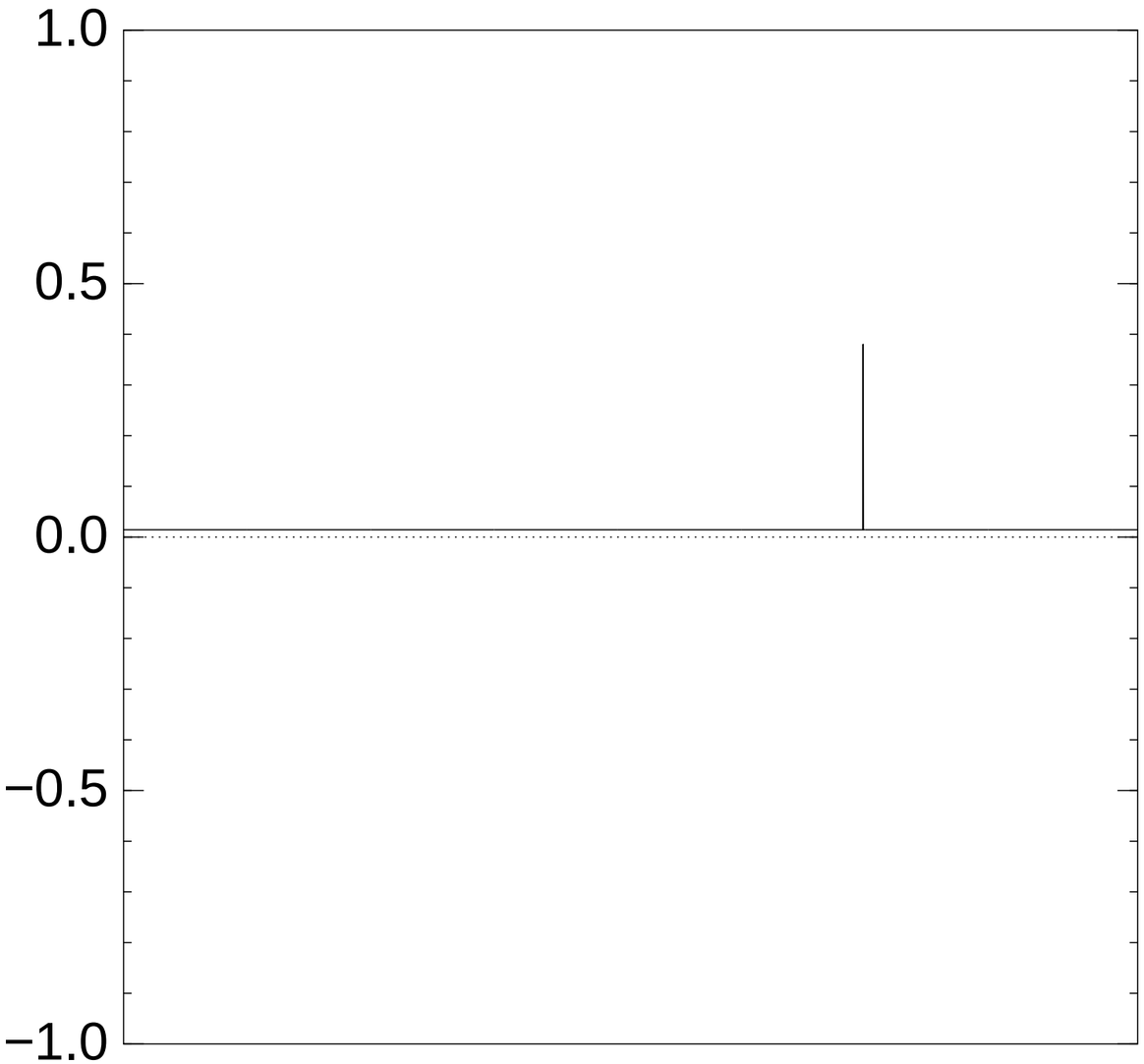
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $12 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

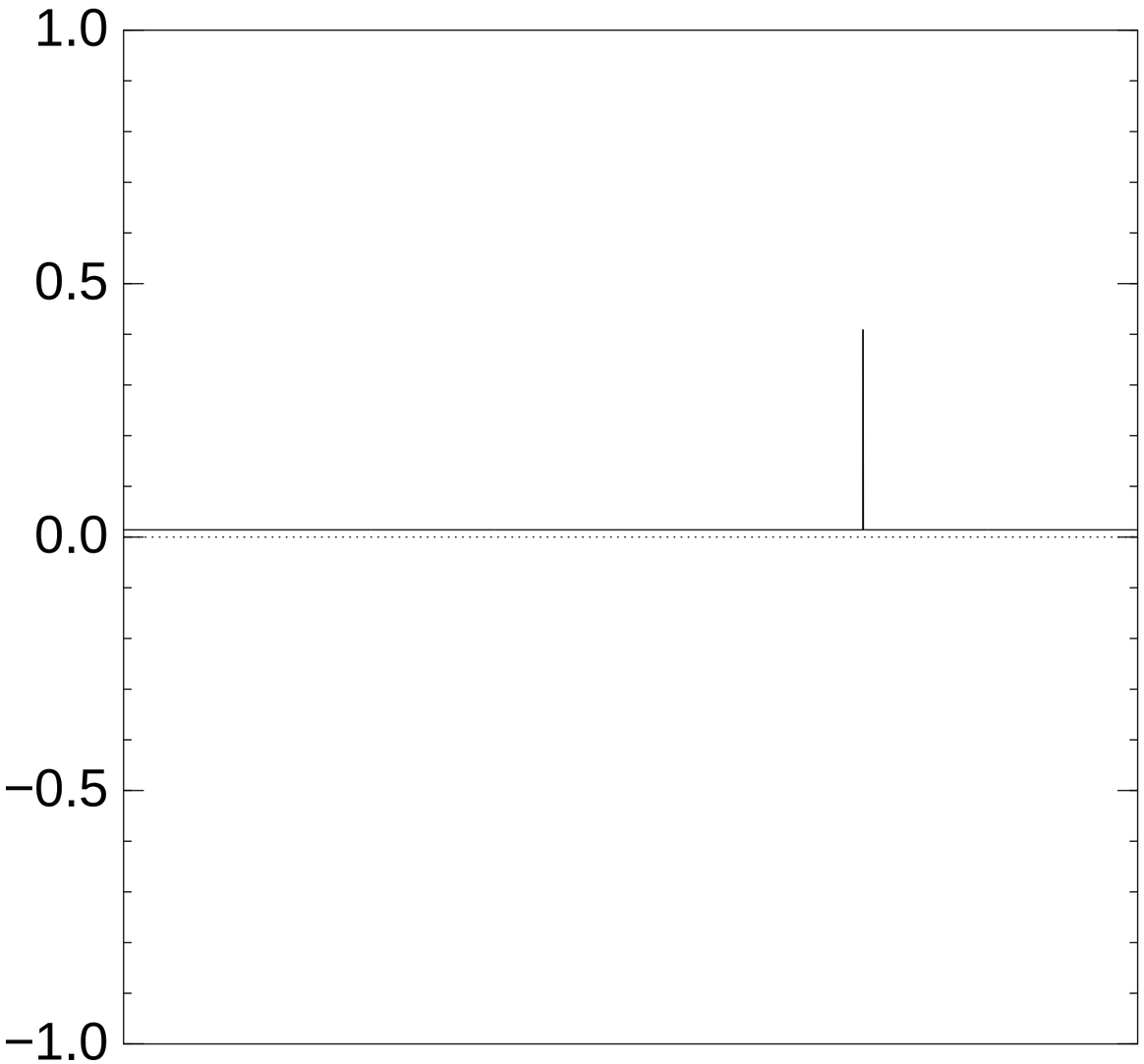
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $13 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

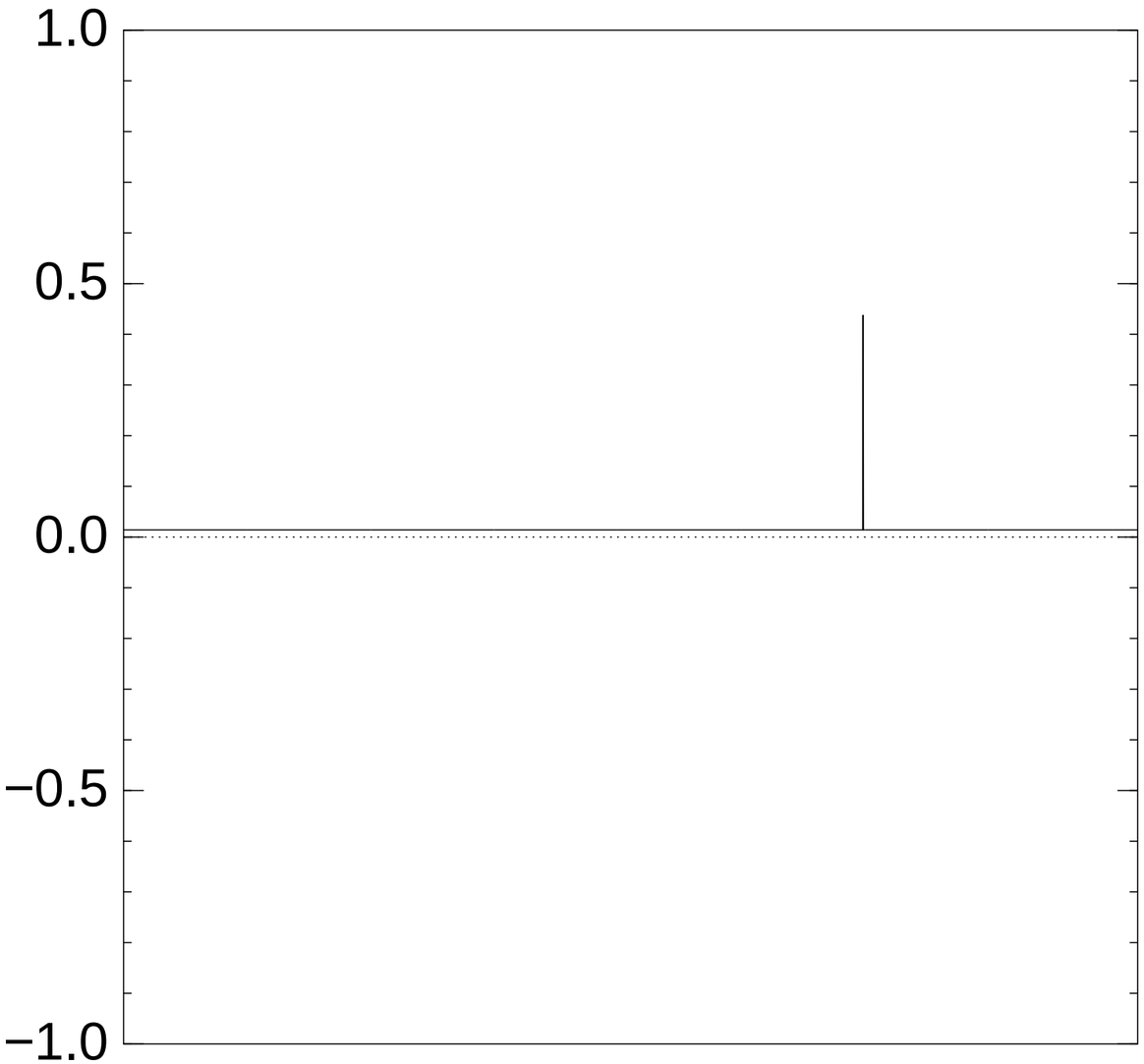
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $14 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

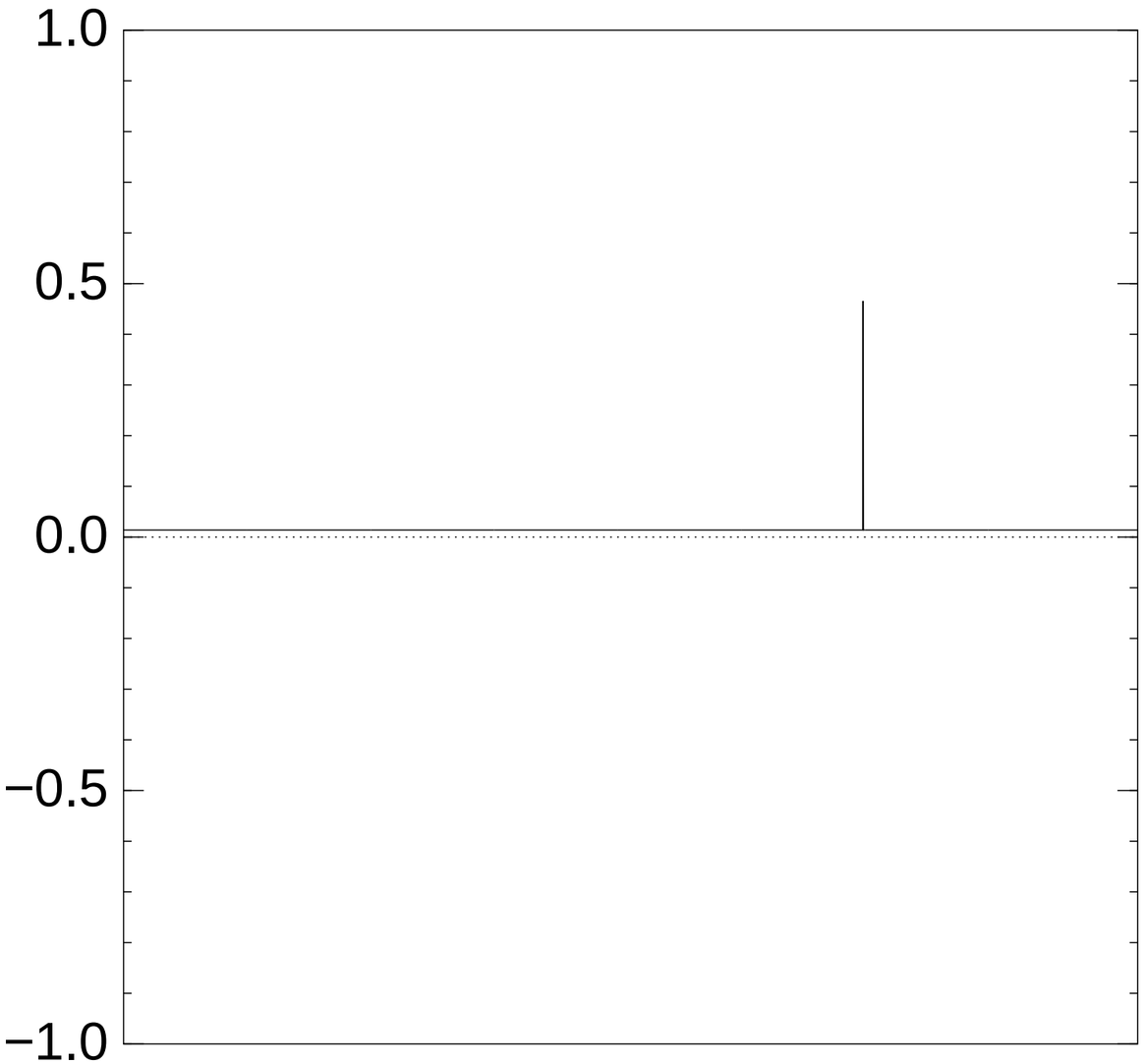
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $15 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

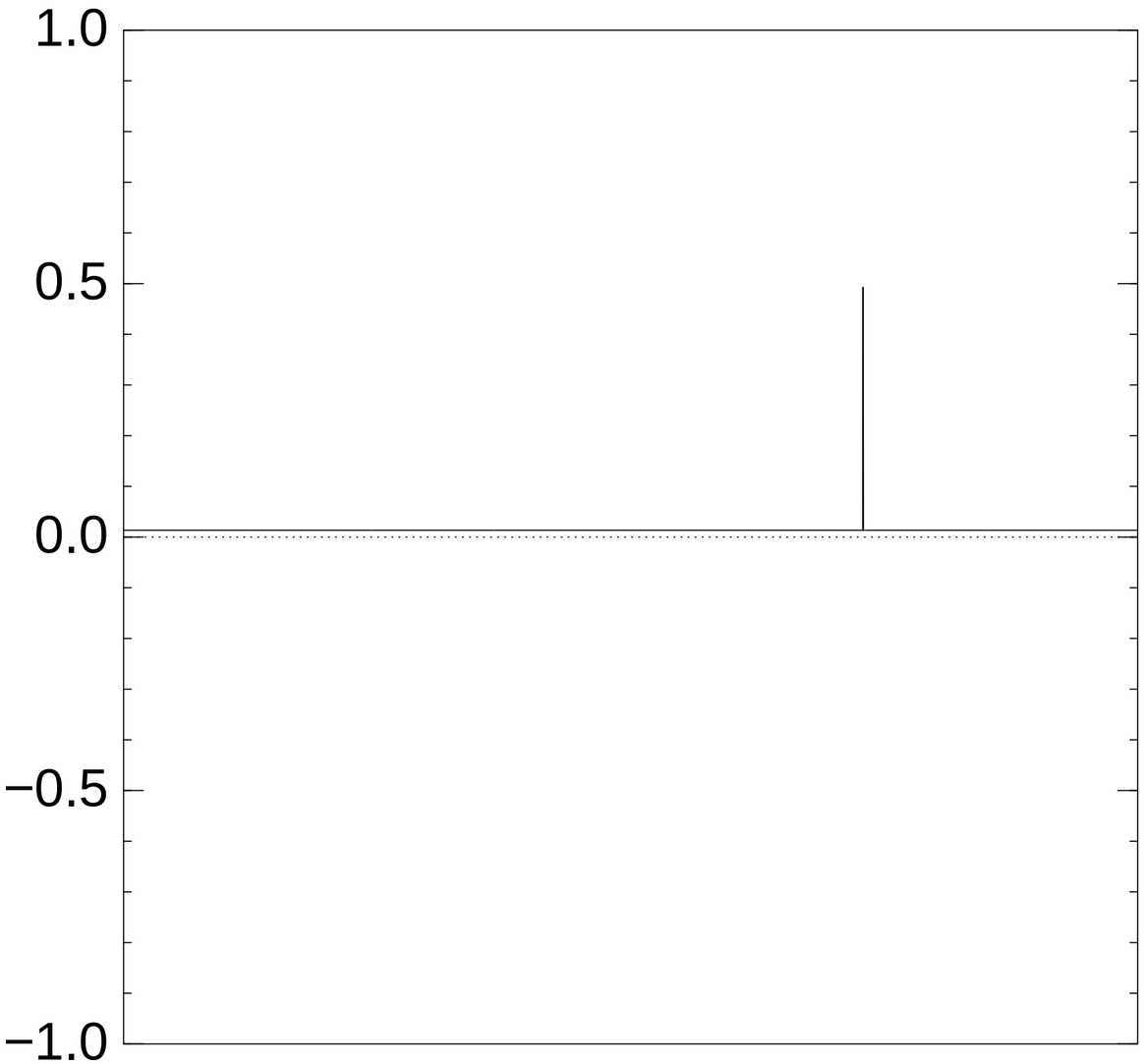
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $16 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

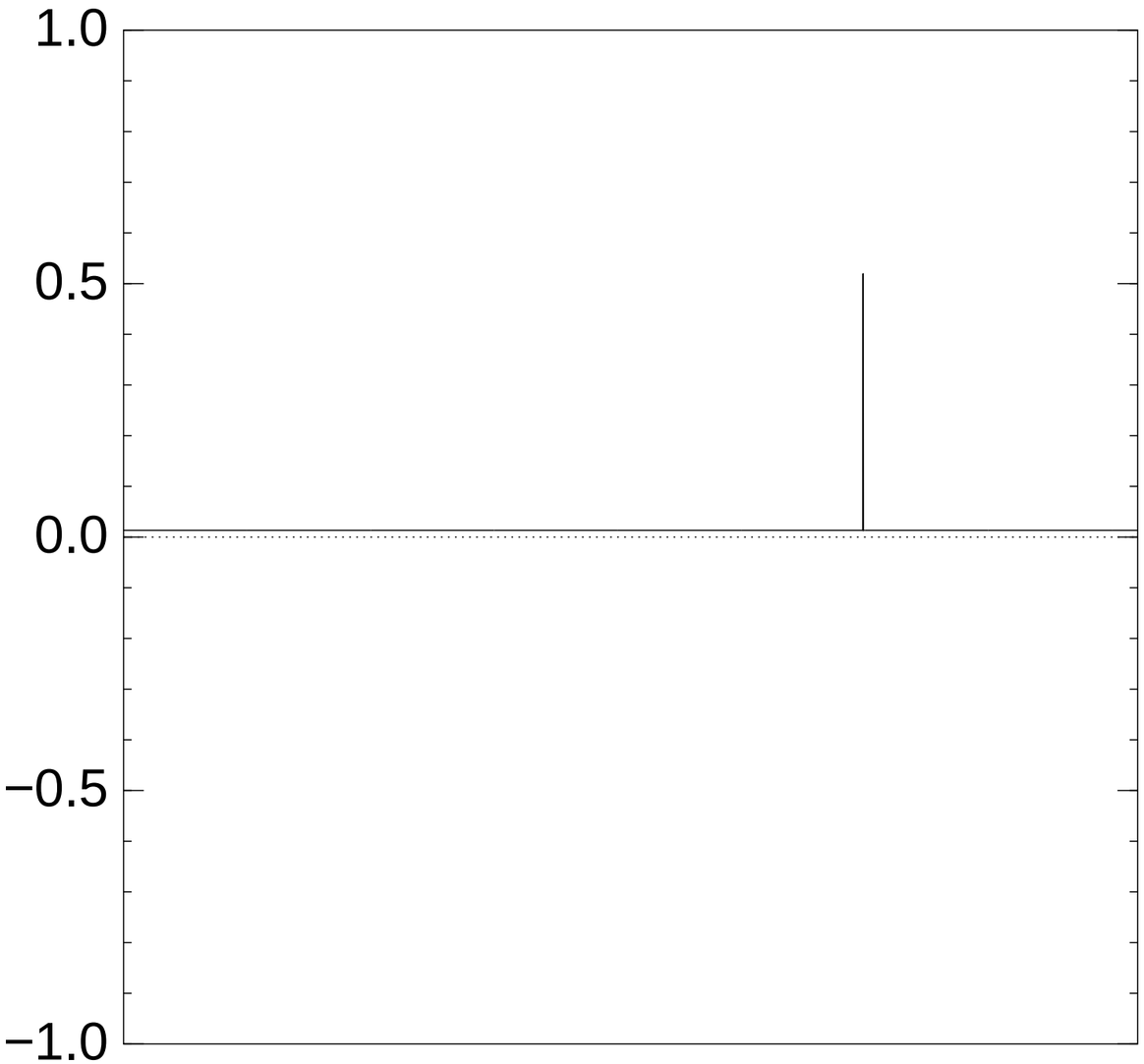
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $17 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

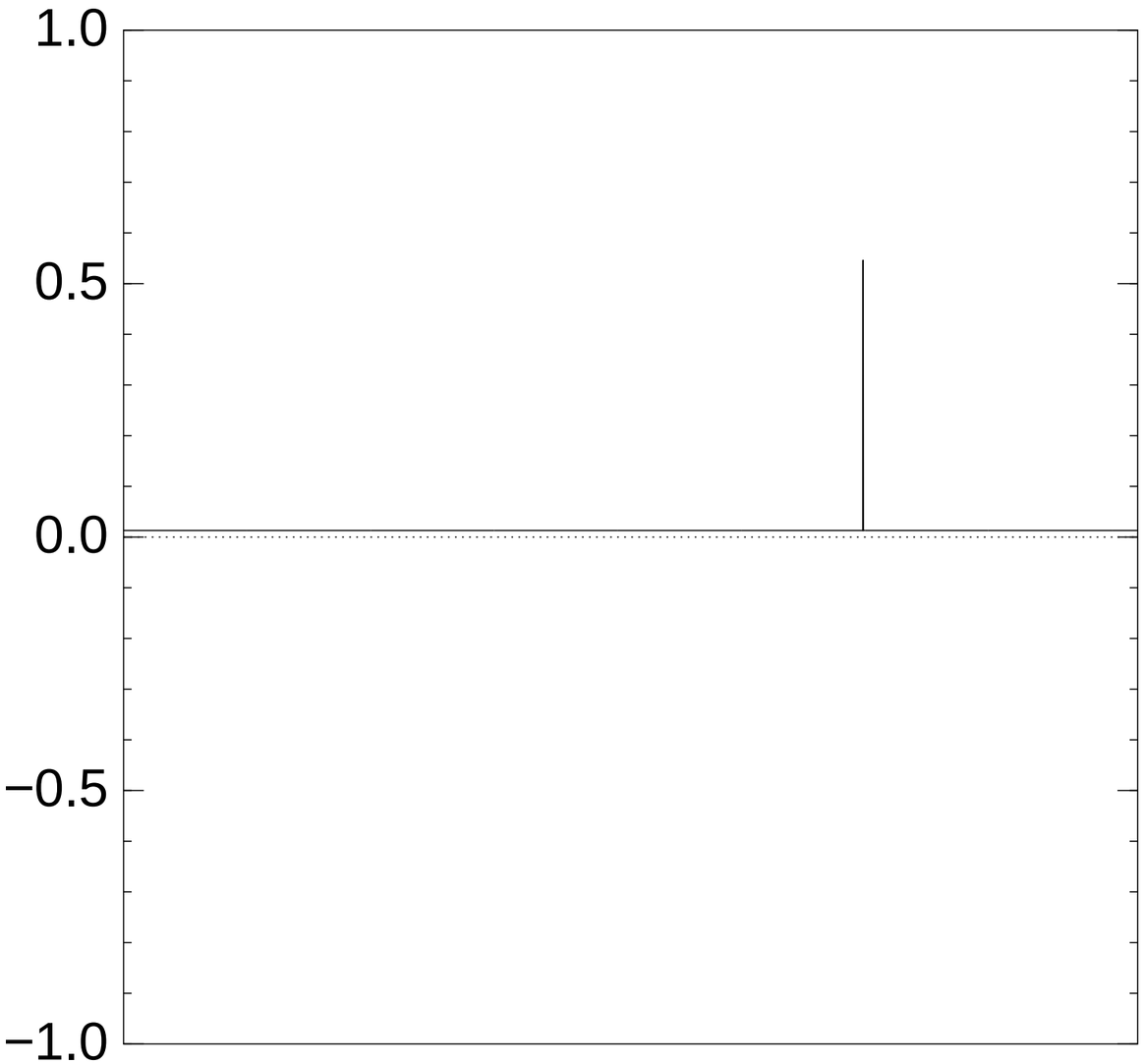
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $18 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

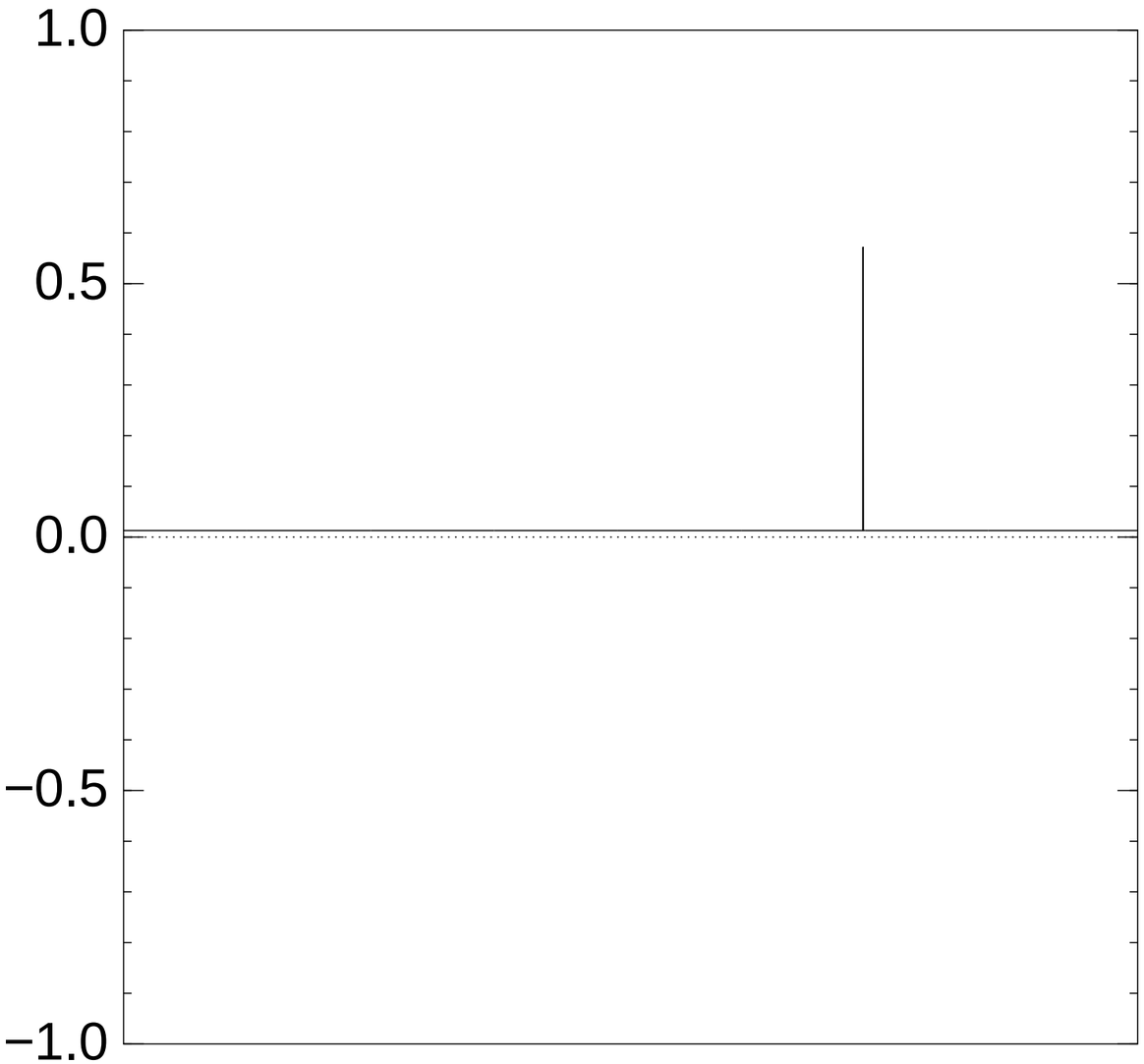
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $19 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

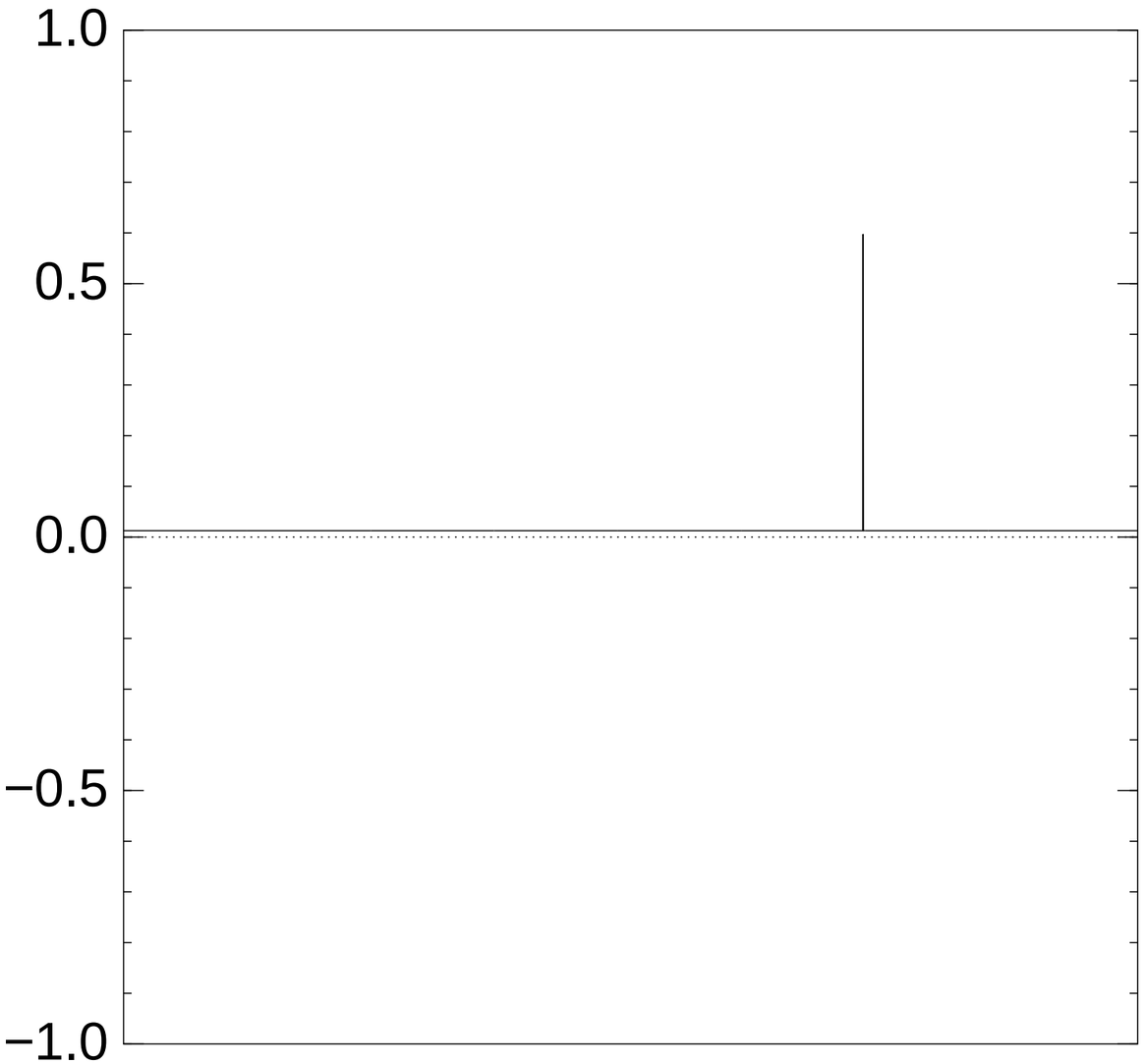
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $20 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

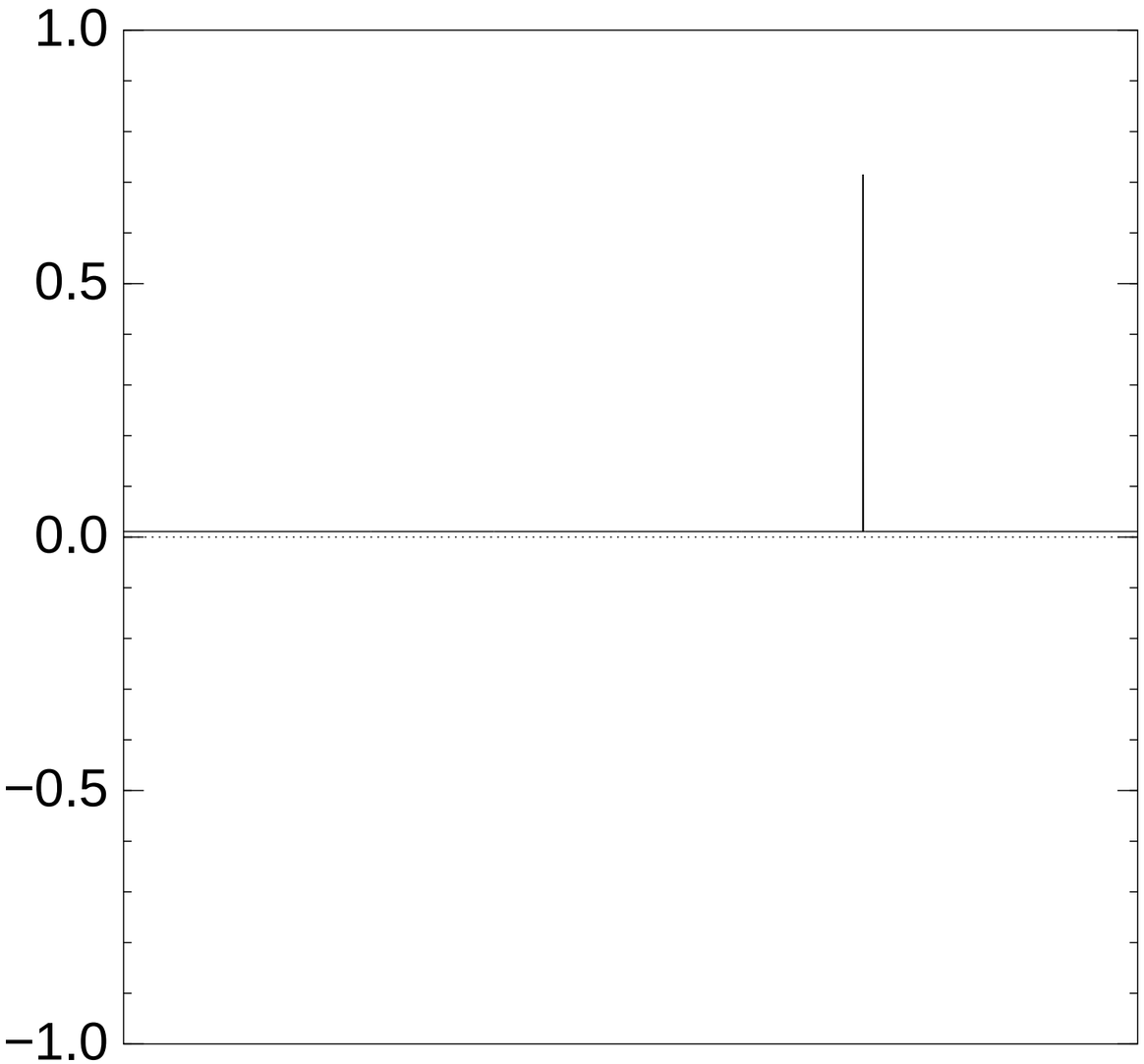
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $25 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

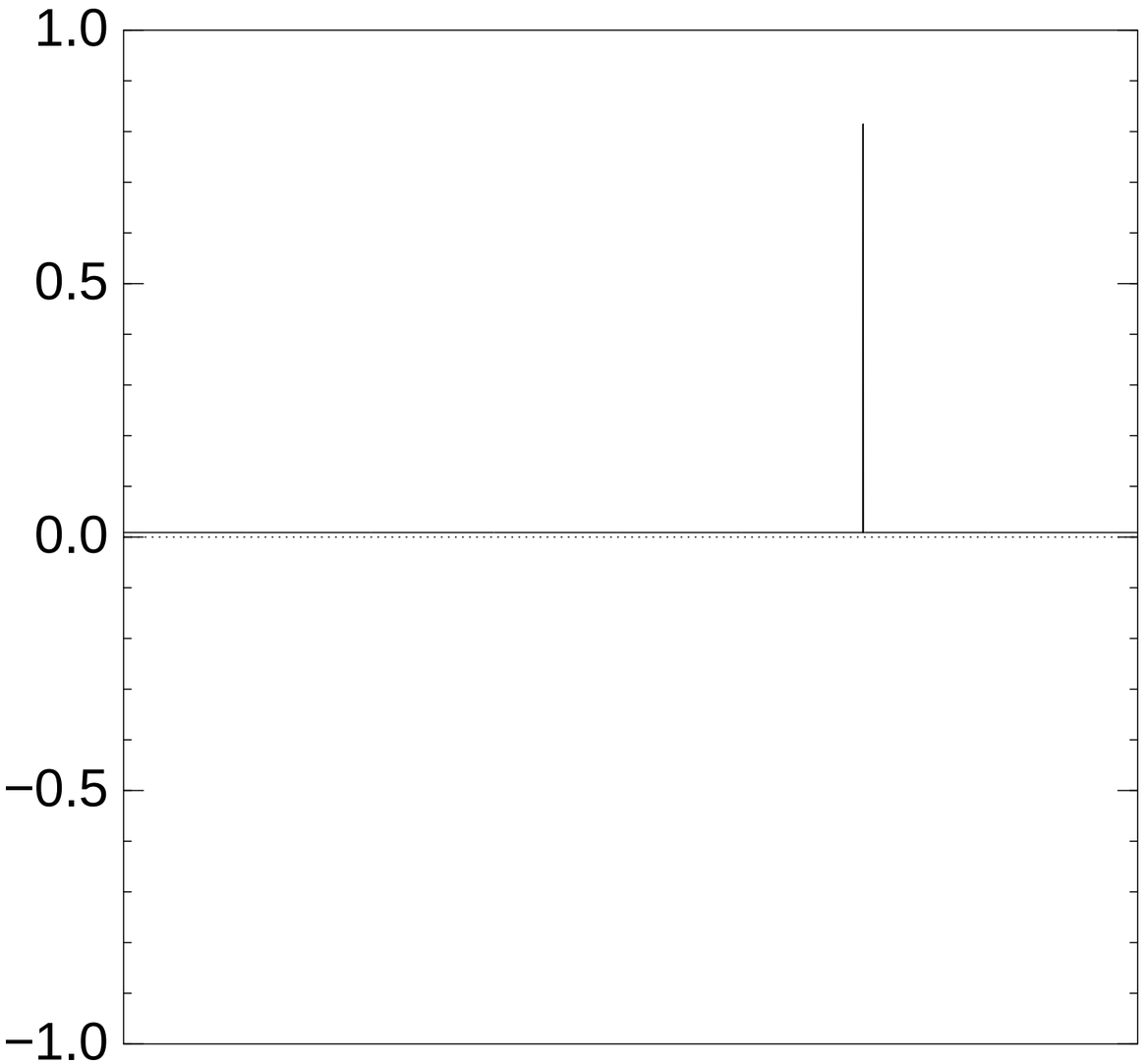
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $30 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

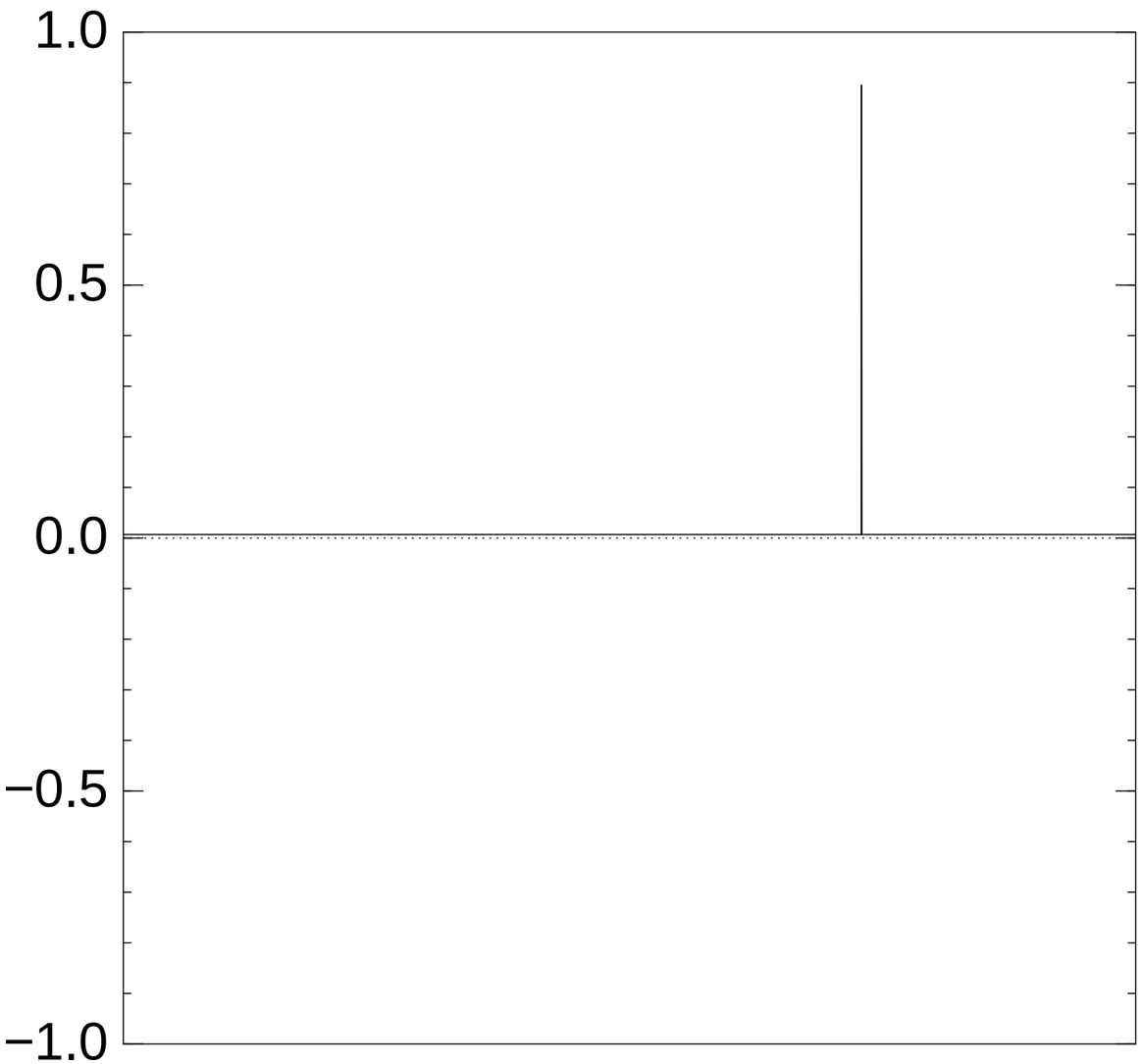
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $35 \times$ (Step 1 + Step 2):



Good moment to stop, measure.

Start from uniform superposition
over all n-bit strings q.

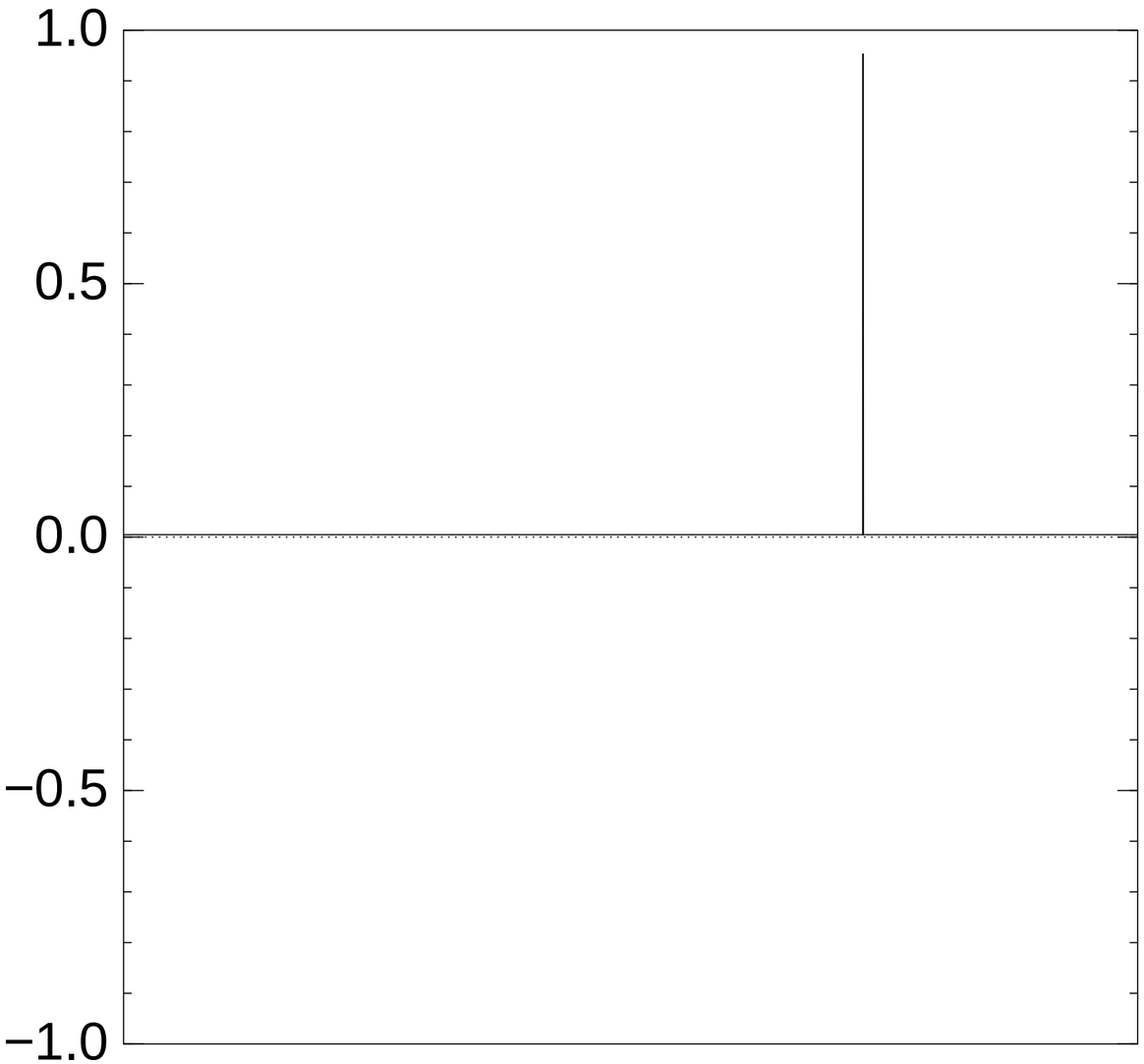
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $40 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

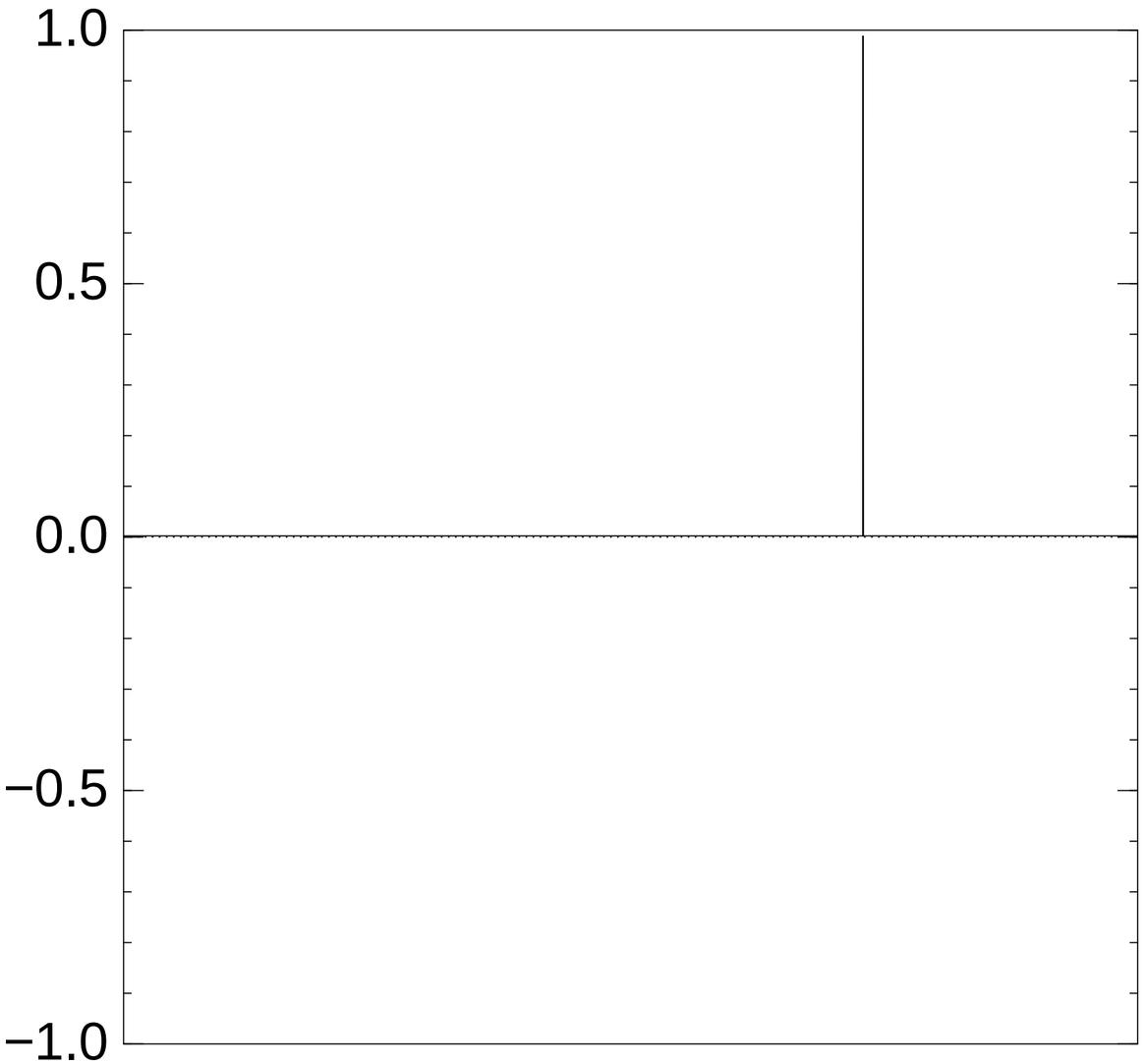
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $45 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

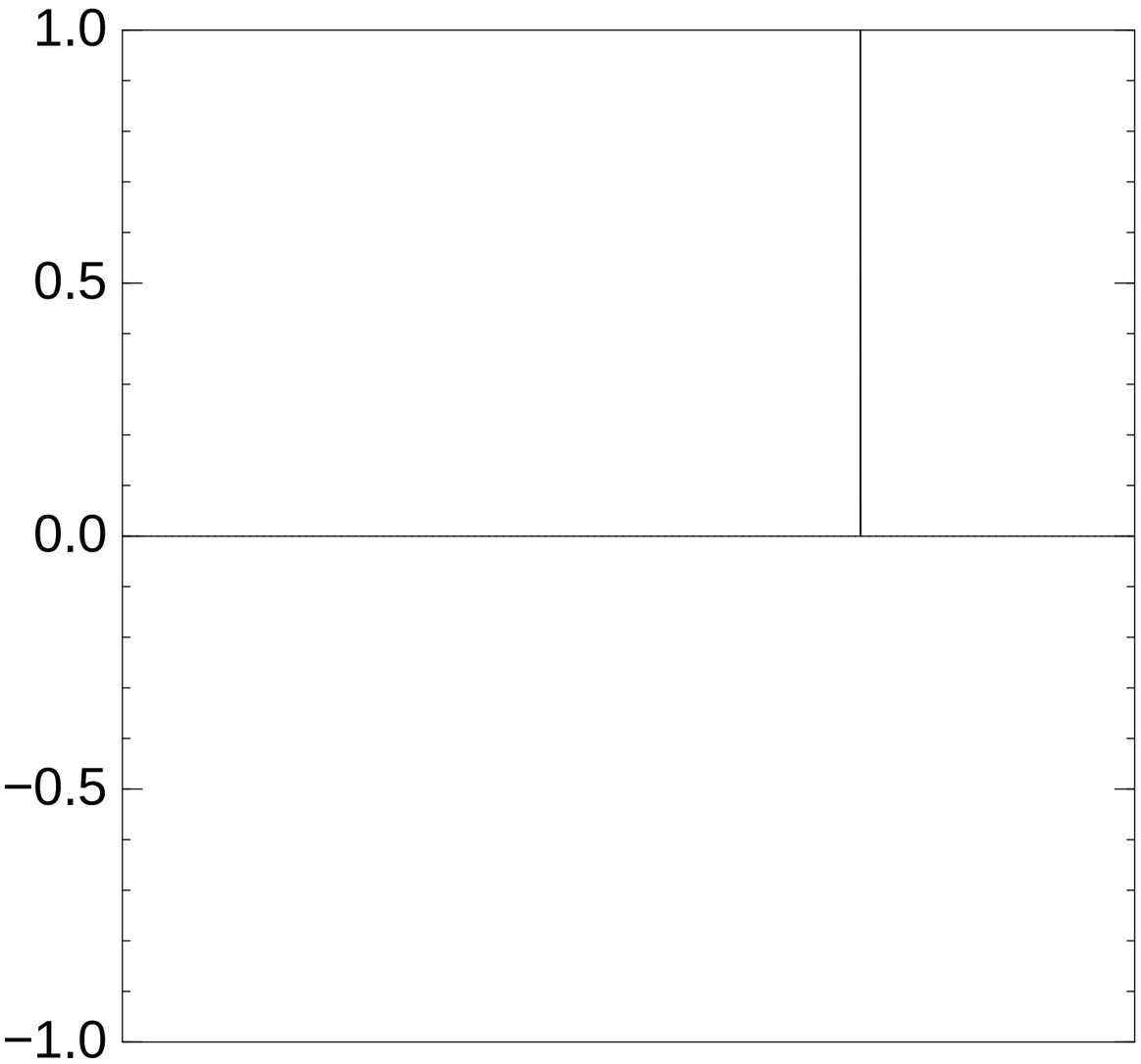
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $50 \times$ (Step 1 + Step 2):



Traditional stopping point.

Start from uniform superposition
over all n-bit strings q.

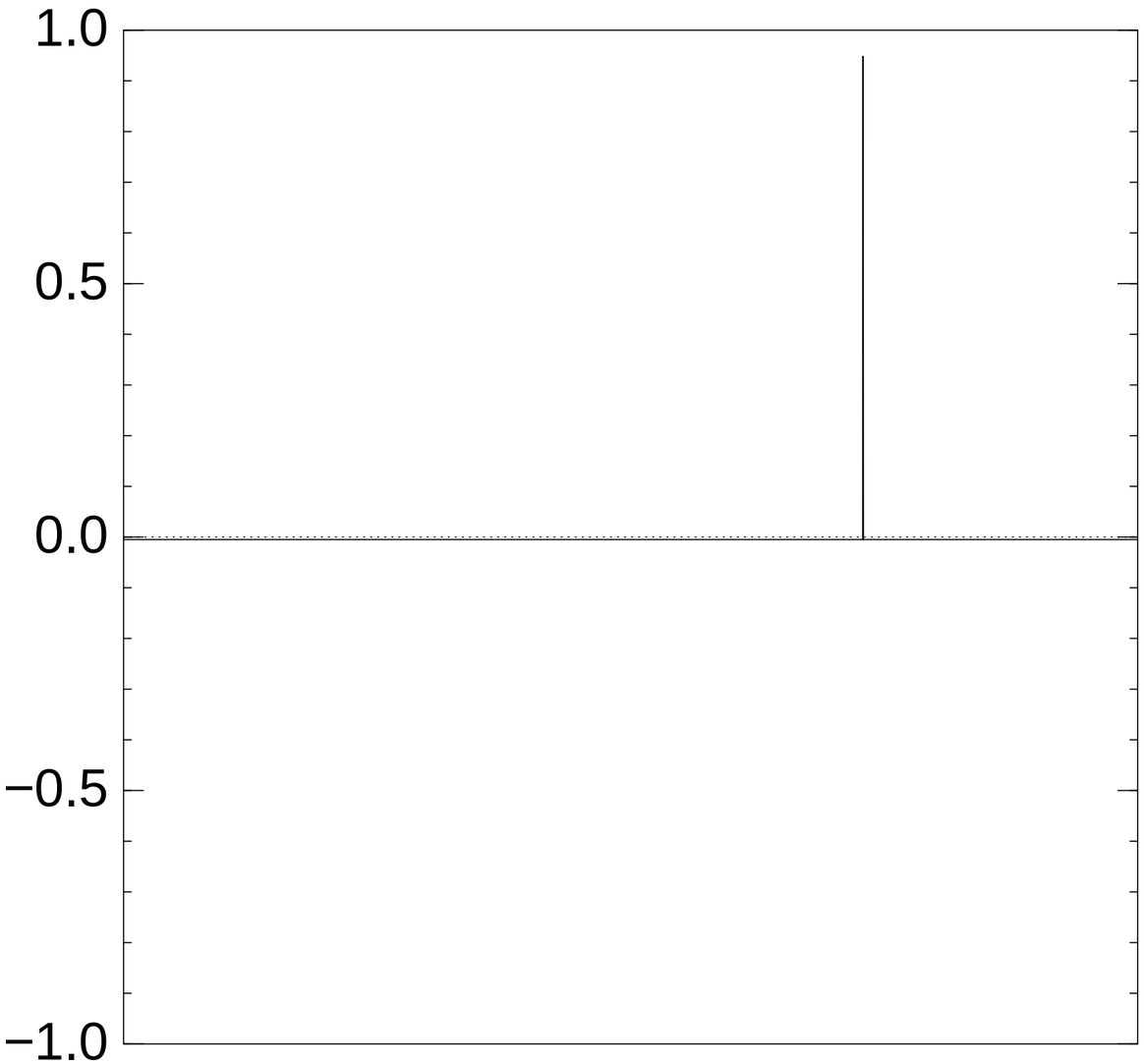
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $60 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

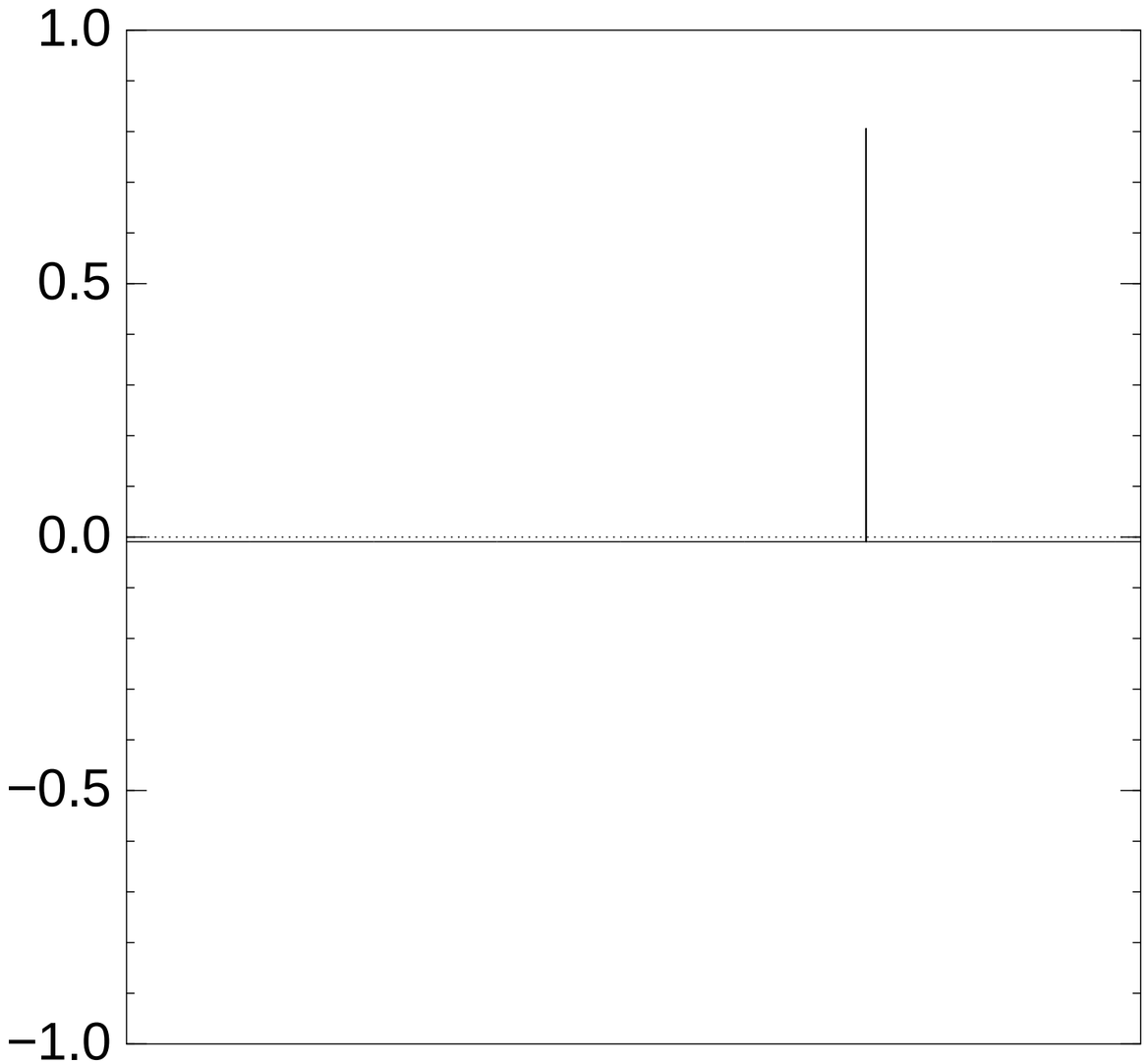
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $70 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

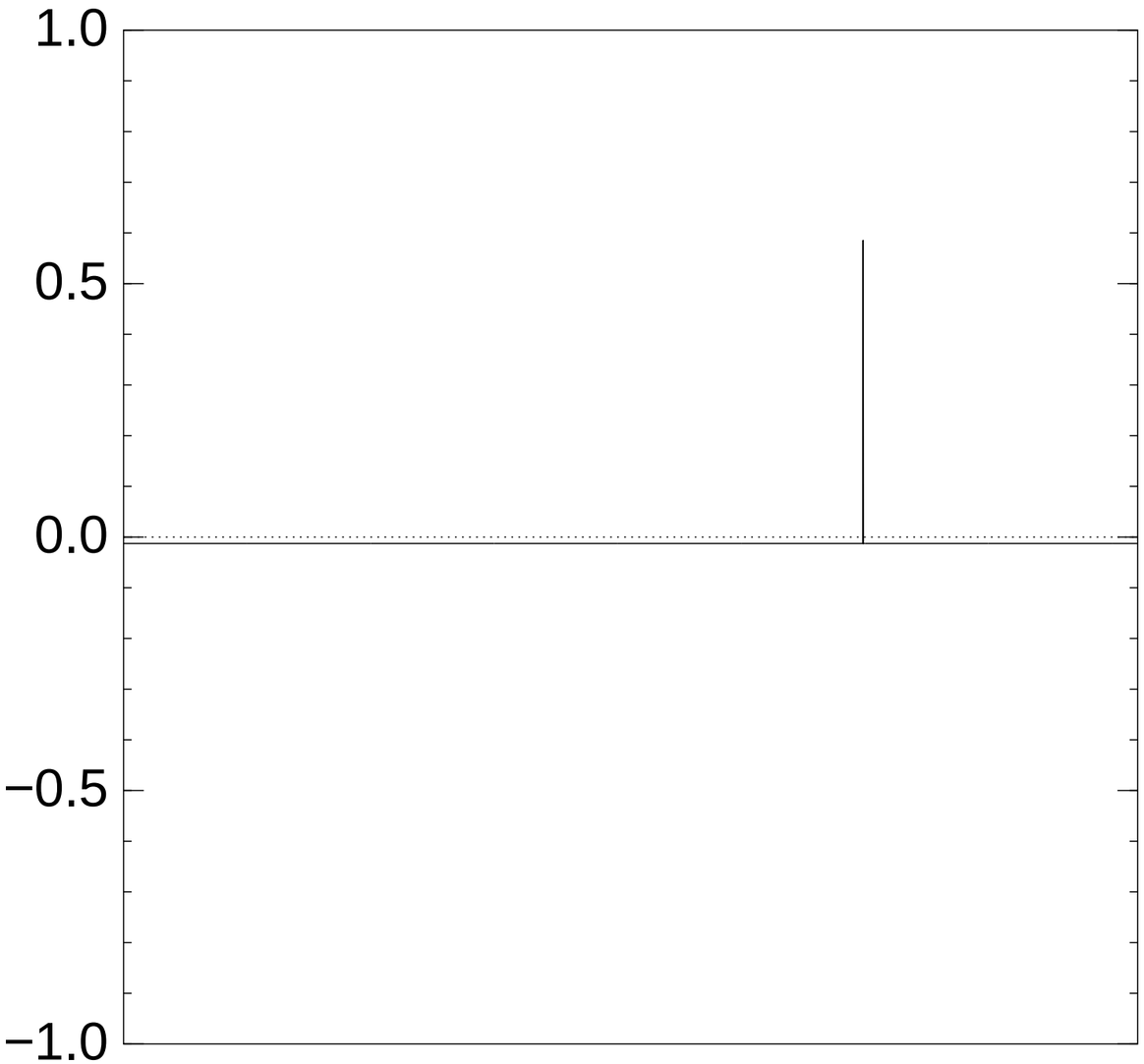
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $80 \times$ (Step 1 + Step 2):



Start from uniform superposition
over all n-bit strings q.

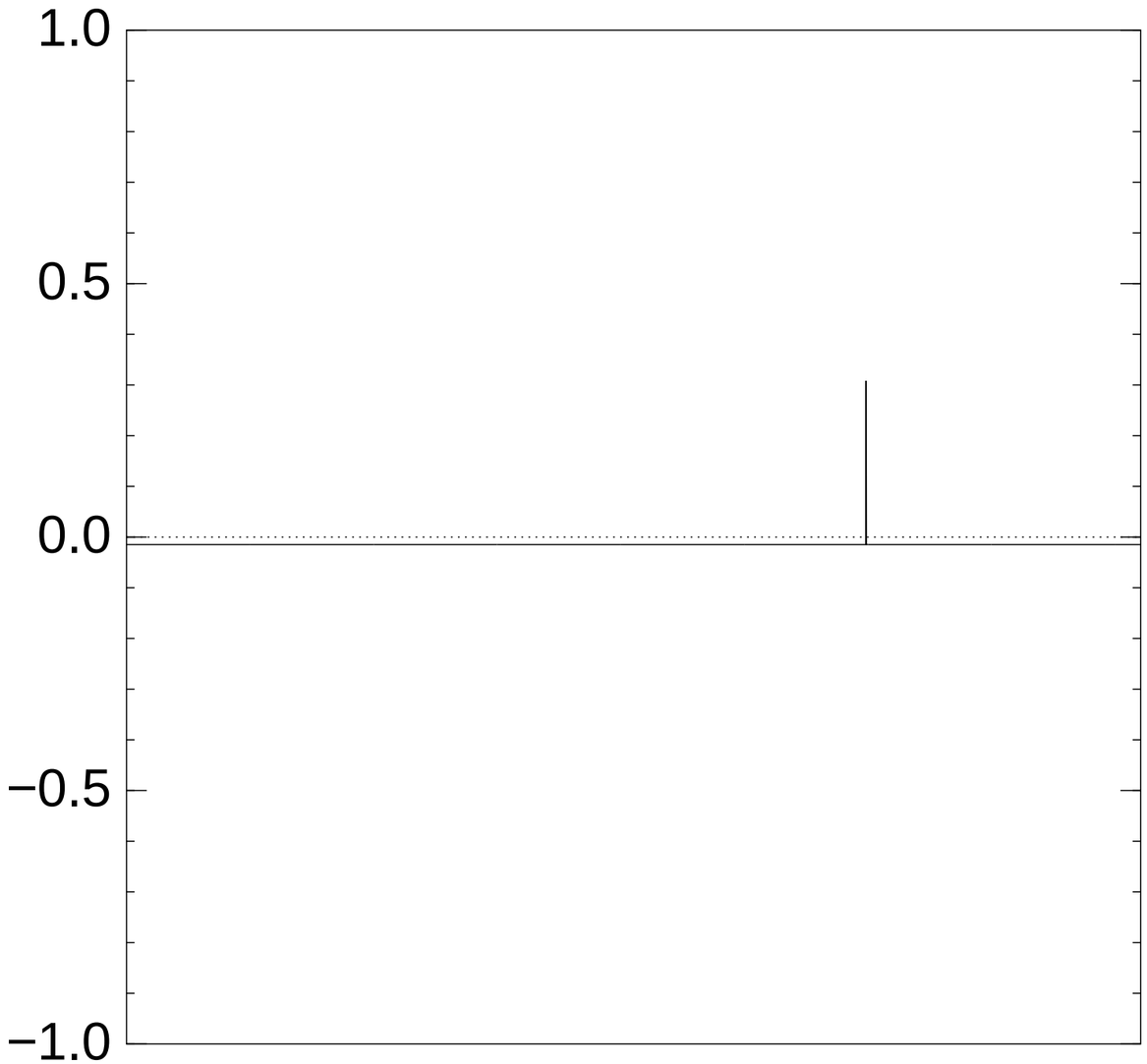
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $90 \times$ (Step 1 + Step 2):



Start from uniform superposition over all n-bit strings q.

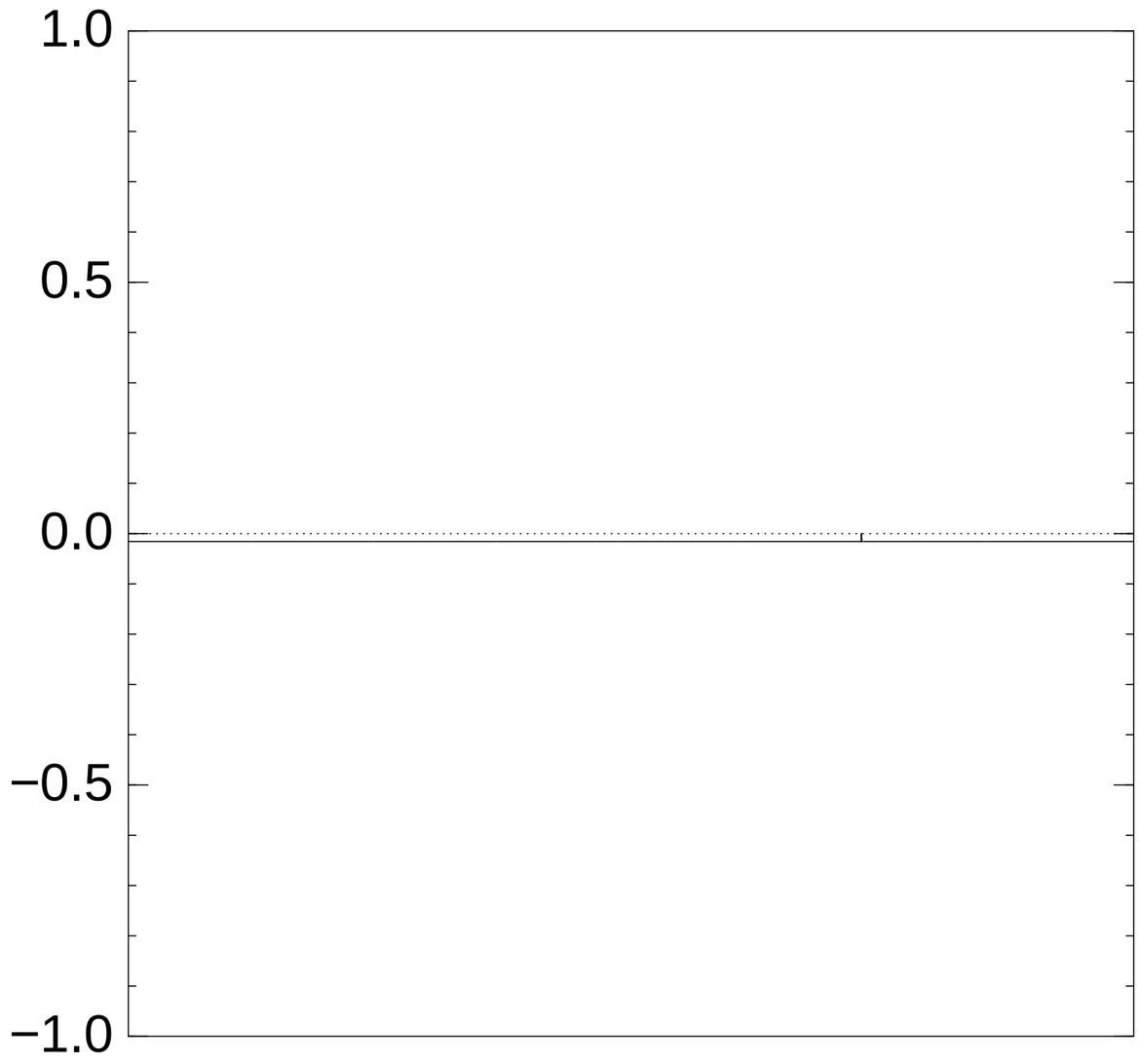
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

Repeat Step 1 + Step 2
about $0.58 \cdot 2^{0.5n}$ 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 $100 \times$ (Step 1 + Step 2):



Very bad stopping point.

om uniform superposition
n-bit strings q.

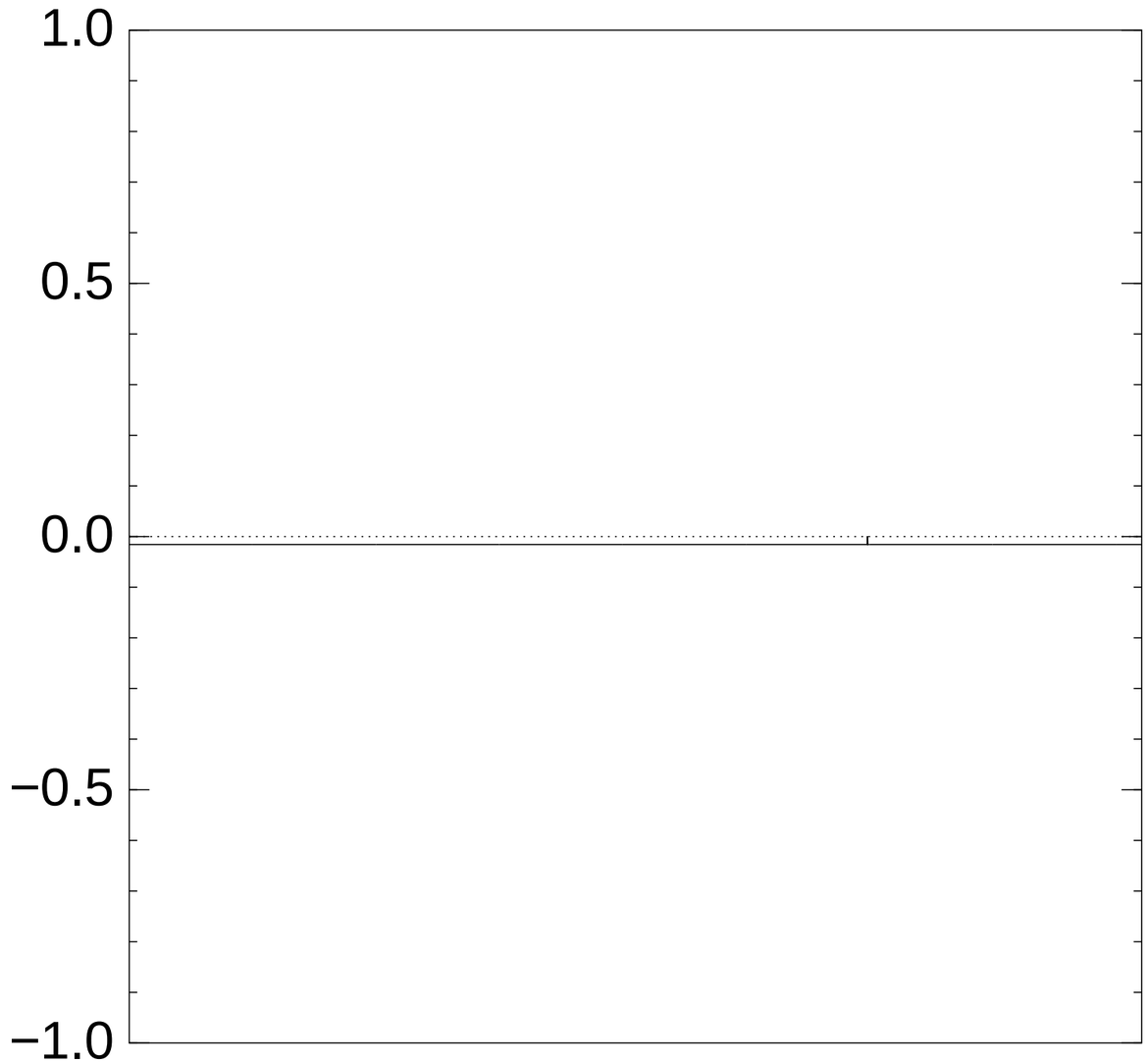
Set $a \leftarrow b$ where
a_q if $f(q) = 0$,
otherwise.
fast.

"Grover diffusion".
a around its average.
also fast.

Step 1 + Step 2
$58 \cdot 2^{0.5n}$ times.

the n qubits.
gh probability this finds s.

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



Very bad stopping point.

$q \mapsto a_q$
by a vec
(with fix
(1) a_q fo
(2) a_q fo

Step 1 +
act linea

Easily co
and pow
to under
of state
$\Rightarrow$ Proba
after $\approx$(

n superposition

gs q.

p where

$= 0$,

.

iffusion".

ts average.

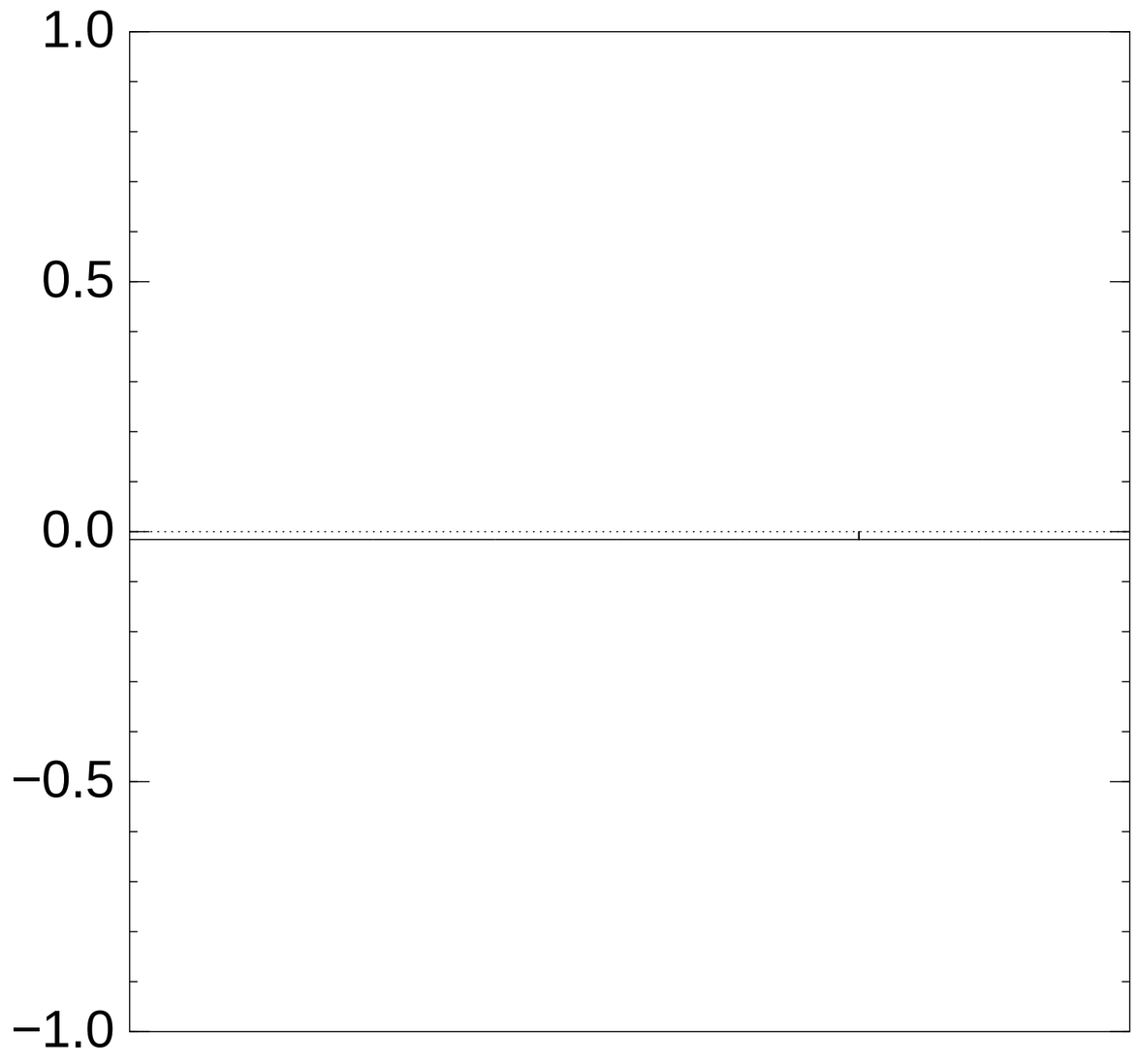
Step 2

times.

bits.

lity this finds s.

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



Very bad stopping point.

$q \mapsto a_q$ is complet
by a vector of two
(with fixed multipl
(1) a_q for roots q;
(2) a_q for non-roo

Step 1 + Step 2
act linearly on this

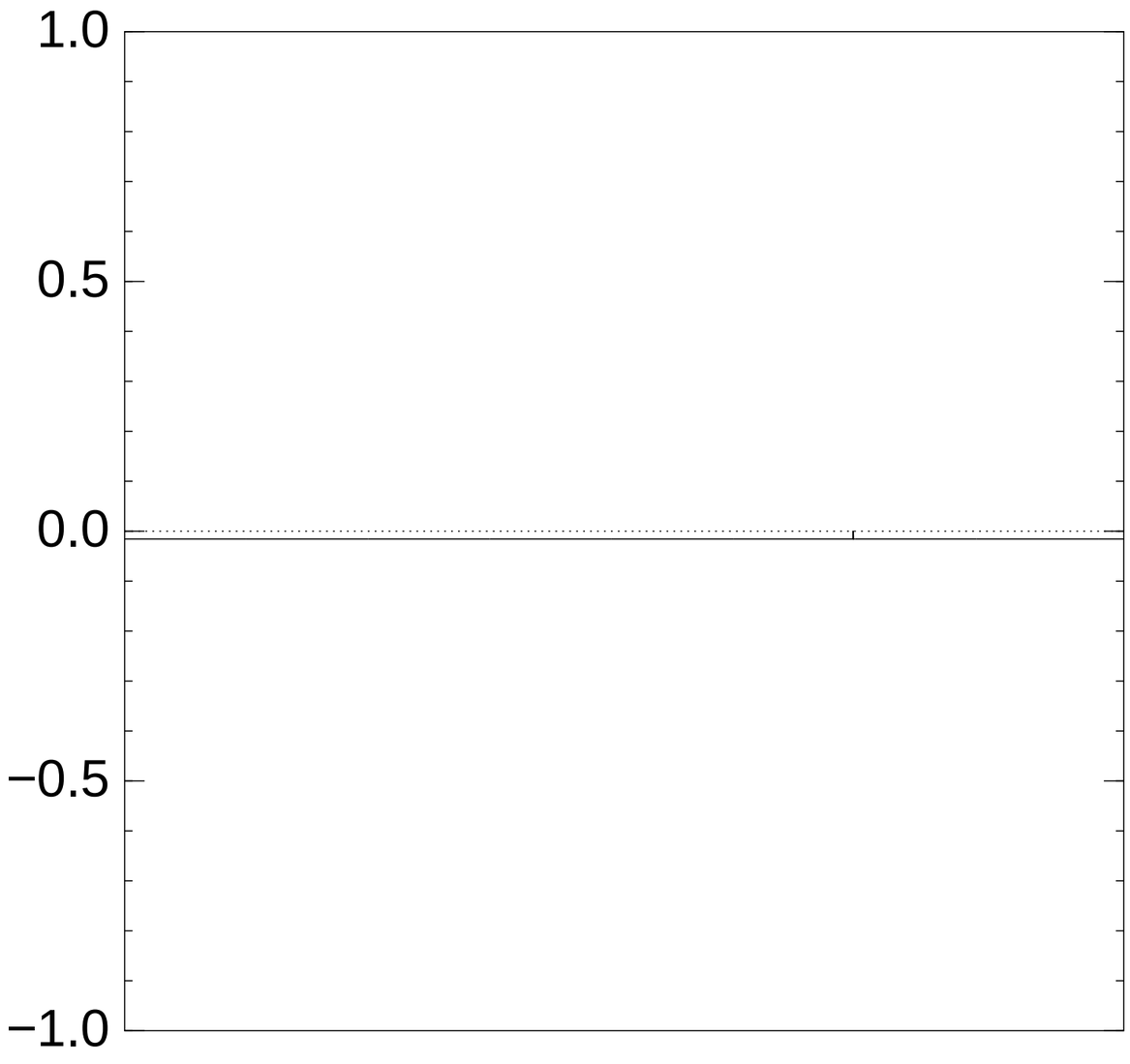
Easily compute eig
and powers of this
to understand evo
of state of Grover'
$\Rightarrow$ Probability is $\approx$
after $\approx(\pi/4)2^{0.5n}$

sition

.

nds s.

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



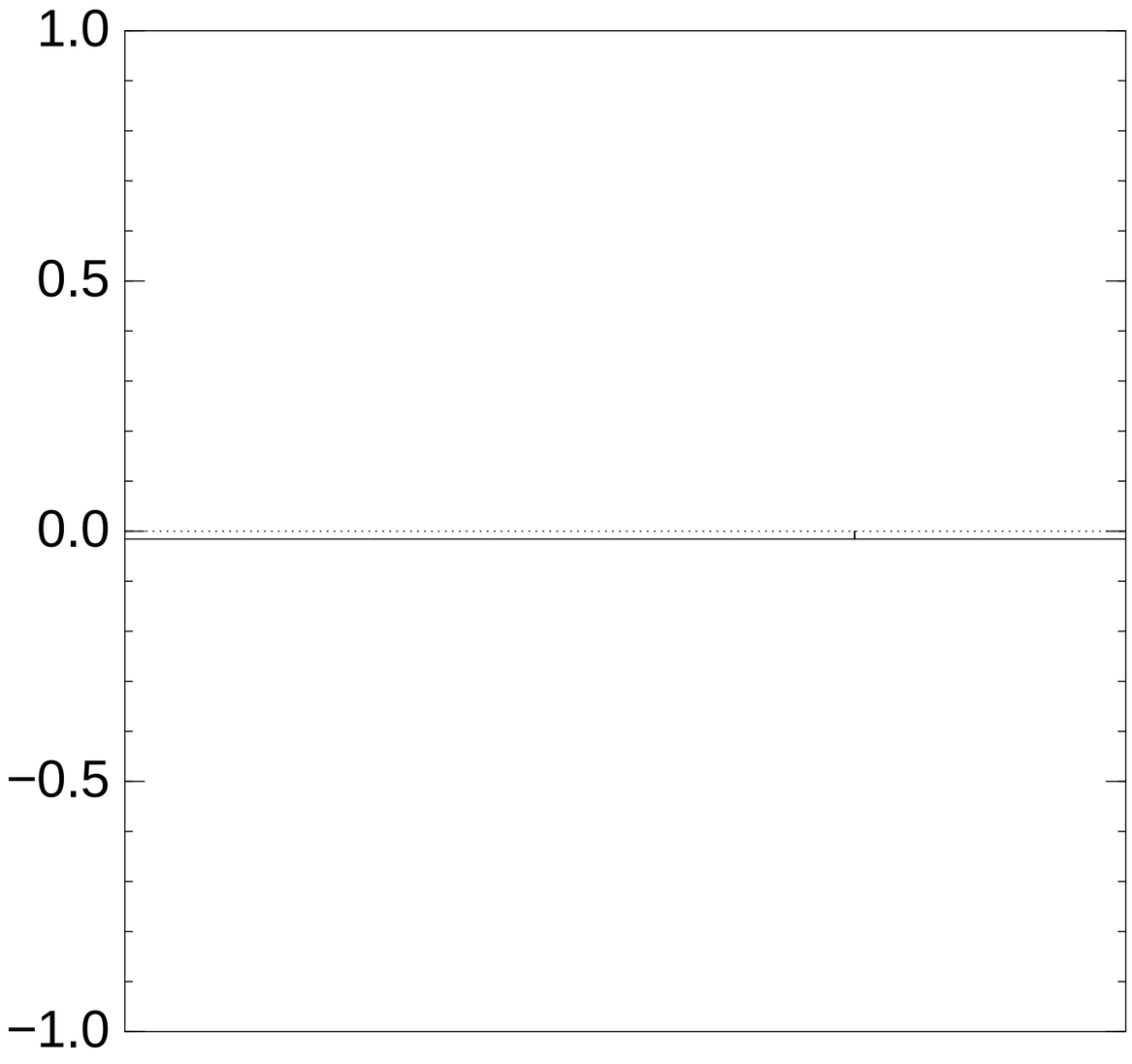
Very bad stopping point.

$q \mapsto a_q$ is completely describ
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 ma
to understand evolution
of state of Grover's algorithm
$\Rightarrow$ Probability is ≈ 1
after $\approx(\pi/4)2^{0.5n}$ iterations

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



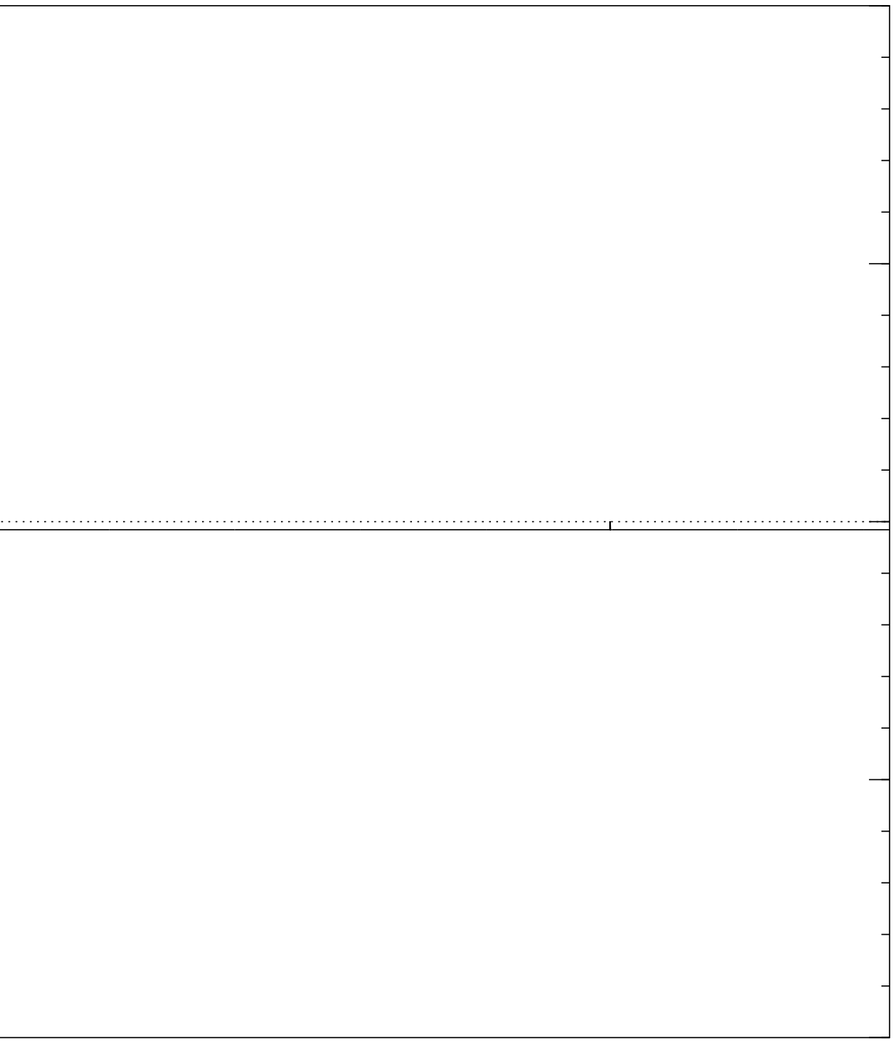
Very bad stopping point.

$q \mapsto a_q$ 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.

Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover's algorithm.
$\Rightarrow$ Probability is ≈ 1
after $\approx (\pi/4)2^{0.5n}$ iterations.

zed graph of $q \mapsto a_q$
xample with $n = 12$
$0 \times$ (Step 1 + Step 2):



d 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.

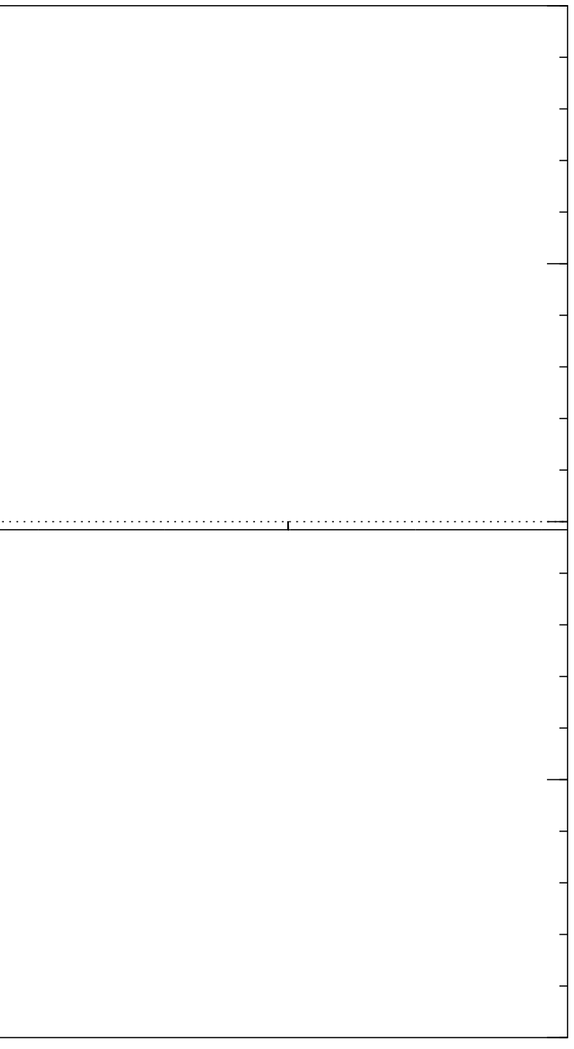
Step 1 + Step 2
act linearly on this vector.

Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover's algorithm.
$\Rightarrow$ Probability is ≈ 1
after $\approx (\pi/4) 2^{0.5n}$ iterations.

Textboo

"WHAT

of $q \mapsto a_q$
th $n = 12$
$1 +$ Step 2):



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.

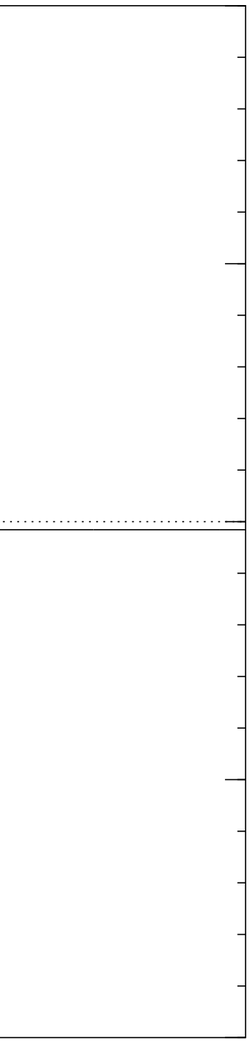
Easily compute eigenvalues
and powers of this linear map
to understand evolution
of state of Grover's algorithm.
$\Rightarrow$ Probability is ≈ 1
after $\approx (\pi/4) 2^{0.5n}$ iterations.

Textbook algorithm

"WHAT is your al

q

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.

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 ≈ 1
after $\approx(\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

$q \mapsto a_q$ 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 ≈ 1
after $\approx (\pi/4) 2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

$q \mapsto a_q$ 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 ≈ 1
after $\approx (\pi/4) 2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

$q \mapsto a_q$ 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 ≈ 1
after $\approx(\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

$q \mapsto a_q$ 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 ≈ 1
after $\approx(\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

$q \mapsto a_q$ 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 ≈ 1
after $\approx (\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

$q \mapsto a_q$ 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 ≈ 1
after $\approx (\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

$q \mapsto a_q$ 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 ≈ 1
after $\approx (\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

is completely described
tor of two numbers
ked multiplicities):
or roots q;
or non-roots q.

+ Step 2
rly on this vector.

ompute eigenvalues
vers of this linear map
rstand evolution
of Grover's algorithm.
ability is ≈ 1
$\pi/4)2^{0.5n}$ iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorith

Critical
How har

tely described
numbers
licities):

ts q.

s vector.

genvalues
s linear map
lution
s algorithm.
≈ 1
iterations.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to atta

Critical question fo
How hard is ECDL

ped

ap

m.

.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to attack crypto

Critical question for ECC se

How hard is ECDLP?

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place.
Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

Textbook algorithm analysis

"WHAT is your algorithm?"

"Heapsort. Here's the code."

"WHAT does it accomplish?"

"It sorts the input array in place. Here's a proof."

"WHAT is its run time?"

"$O(n \lg n)$ comparisons;
and $\Theta(n \lg n)$ comparisons
for most inputs. Here's a proof."

"You may pass."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

k algorithm analysis

is your algorithm?"

ort. Here's the code."

does it accomplish?"

the input array in place.

proof."

is its run time?"

n) comparisons;

$n \lg n$) comparisons

t inputs. Here's a proof."

ay pass."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Ga
inadequa
of a neg

m analysis

gorithm?"

the code."

ccomplish?"

array in place.

time?"

isons;

parisons

Here's a proof."

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lam
inadequately speci
of a negating rho

”

”

place.

oof.”

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for “strong”
ECC groups of prime order ℓ:
Latest “negating” variants of
“distinguished point” rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it’s true?

2000 Gallant–Lambert–Vans
inadequately specified stater
of a negating rho algorithm.

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Algorithms to attack crypto

Critical question for ECC security:
How hard is ECDLP?

Standard estimate for "strong"
ECC groups of prime order ℓ:
Latest "negating" variants of
"distinguished point" rho methods
break an average ECDLP instance
using $\approx 0.886\sqrt{\ell}$ additions.

Is this proven? No!

Is this provable? Maybe not!

So why do we think it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

ms to attack crypto

question for ECC security:
rd is ECDLP?

d estimate for "strong"
oups of prime order ℓ:
negating" variants of
uished point" rho methods
average ECDLP instance
$0.886\sqrt{\ell}$ additions.

roven? No!

rovable? Maybe not!

do we think it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar s
we don't
best fact

ack crypto

or ECC security:
LP?

for "strong"
me order ℓ:
variants of
nt" rho methods
ECDLP instance
dditions.

o!

Maybe not!

nk it's true?

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for R
we don't have pro
best factoring algo

curity:

ng"
ℓ:
of
ethods
stance

!

e?

31

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

32

Similar story for RSA securit
we don't have proofs for the
best factoring algorithms.

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

2000 Gallant–Lambert–Vanstone:
inadequately specified statement
of a negating rho algorithm.

2010 Bos–Kleinjung–Lenstra:
a plausible interpretation of
that algorithm is *non-functional*.

See 2011 Bernstein–Lange–
Schwabe for more history
and better algorithms.

Why do we believe that
the latest algorithms work
at the claimed speeds?
Experiments!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

llant–Lambert–Vanstone:
ately specified statement
ating rho algorithm.

s–Kleinjung–Lenstra:
ble interpretation of
orithm is *non-functional*.

1 Bernstein–Lange–
e for more history
ter algorithms.

we believe that
st algorithms work
laimed speeds?
nents!

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

Where's

Quantur
is movin
into algo

Example
exponen
Bernstei

Don't ex
for the b
to attac

How do
in analys
Quantur

bert–Vanstone:
fied statement
algorithm.

ng–Lenstra:
etation of
non-functional.

n–Lange–
history
hms.

e that
ms work
eeds?

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

Where's my quant

Quantum-algorithm
is moving beyond
into algorithms wi

Example: subset-s
exponent ≈0.241
Bernstein–Jeffery–

Don't expect proo
for the best quant
to attack post-qua

How do we obtain
in analysis of these
Quantum experim

stone:
ment
n:
onal.

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

Where's my quantum compu

Quantum-algorithm design
is moving beyond textbook s
into algorithms without proo

Example: subset-sum
exponent ≈0.241 from 2013
Bernstein–Jeffery–Lange–Me

Don't expect proofs or prova
for the best quantum algorit
to attack post-quantum cryp

How do we obtain confidenc
in analysis of these algorithm
Quantum experiments are ha

Similar story for RSA security:
we don't have proofs for the
best factoring algorithms.

Code-based cryptography:
we don't have proofs for the
best decoding algorithms.

Lattice-based cryptography:
we don't have proofs for the
best lattice algorithms.

MQ-based cryptography:
we don't have proofs for the
best system-solving algorithms.

Confidence relies on experiments.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent ≈ 0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

story for RSA security:
t have proofs for the
toring algorithms.

sed cryptography:
t have proofs for the
coding algorithms.

based cryptography:
t have proofs for the
tice algorithms.

ed cryptography:
t have proofs for the
tem-solving algorithms.

nce relies on experiments.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's

Analogy
a 2^{80} NF

SA security:
ofs for the
orithms.

ography:
ofs for the
orithms.

otography:
ofs for the
hms.

graphy:
ofs for the
g algorithms.

on experiments.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's my big co

Analogy: Public h
a 2^{80} NFS RSA-10

ty:

e

e

e

ms.

nents.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's my big computer?

Analogy: Public hasn't carri
a 2^{80} NFS RSA-1024 experi

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's my big computer?

Analogy: Public hasn't carried out
a 2^{80} NFS RSA-1024 experiment.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's my big computer?

Analogy: Public hasn't carried out
a 2^{80} NFS RSA-1024 experiment.

But public has carried out
2^{50}, 2^{60}, 2^{70} NFS experiments.
Hopefully not too much
extrapolation error for 2^{80}.

Where's my quantum computer?

Quantum-algorithm design
is moving beyond textbook stage
into algorithms without proofs.

Example: subset-sum
exponent $\approx$0.241 from 2013
Bernstein–Jeffery–Lange–Meurer.

Don't expect proofs or provability
for the best quantum algorithms
to attack post-quantum crypto.

How do we obtain confidence
in analysis of these algorithms?
Quantum experiments are hard.

Where's my big computer?

Analogy: Public hasn't carried out
a 2^{80} NFS RSA-1024 experiment.

But public has carried out
2^{50}, 2^{60}, 2^{70} NFS experiments.
Hopefully not too much
extrapolation error for 2^{80}.

Vastly larger extrapolation
for the quantum situation.
Imagine attacker performing
2^{80} operations on 2^{40} qubits;
compare to today's challenges
of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

my quantum computer?

m-algorithm design
g beyond textbook stage
orithms without proofs.

e: subset-sum
t $\approx$0.241 from 2013
n–Jeffery–Lange–Meurer.

xpect proofs or provability
best quantum algorithms
k post-quantum crypto.

we obtain confidence
sis of these algorithms?
m experiments are hard.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments.
Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation.
Imagine attacker performing 2^{80} operations on 2^{40} qubits;
compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulati

2014.04
Simulati
proof of
distinctn

tum computer?

m design
textbook stage
thout proofs.

sum
from 2013
-Lange–Meurer.

fs or provability
um algorithms
antum crypto.

confidence
e algorithms?
ents are hard.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments.
Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation.
Imagine attacker performing 2^{80} operations on 2^{40} qubits;
compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou →
Simulation shows
proof of 2003 Am
distinctness algorit

uter?

stage

ofs.

eurer.

ability

thms

pto.

ce

ms?

ard.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments.
Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation.
Imagine attacker performing 2^{80} operations on 2^{40} qubits; compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments. Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing 2^{80} operations on 2^{40} qubits; compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments. Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing 2^{80} operations on 2^{40} qubits; compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou $\rightarrow$ Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments. Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing 2^{80} operations on 2^{40} qubits; compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Where's my big computer?

Analogy: Public hasn't carried out a 2^{80} NFS RSA-1024 experiment.

But public has carried out 2^{50}, 2^{60}, 2^{70} NFS experiments. Hopefully not too much extrapolation error for 2^{80}.

Vastly larger extrapolation for the quantum situation. Imagine attacker performing 2^{80} operations on 2^{40} qubits; compare to today's challenges of 2^1, 2^2, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.

my big computer?

: Public hasn't carried out
FS RSA-1024 experiment.

lic has carried out
, 2^{70} NFS experiments.
y not too much
ation error for 2^{80}.

arger extrapolation
quantum situation.
attacker performing
rations on 2^{40} qubits;
e to today's challenges
^{2}, 2^3, 2^4, 2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs:
Simulation shows that 2003
Childs–Eisenberg distinctness
algorithm is non-functional;
need to take half angle.

Childs: Yes. Typo, already
fixed in 2005 journal version.

Do we k

Maybe,

How ma
looked f

omputer?

asn't carried out
024 experiment.

rried out
experiments.
much
r for 2^{80}.

polation
ituation.
performing
2^{40} qubits;
s challenges
2^5, 2^6 qubits.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.

Do we know the b

Maybe, maybe not

How many researc
looked for better a

ed out
ment.

nts.

;
s;
es
its.

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.

Do we know the best attack

Maybe, maybe not.

How many researchers have looked for better attacks?

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.

Do we know the best attacks?

Maybe, maybe not.

How many researchers have looked for better attacks?

Simulations

2014.04 Chou → Ambainis: Simulation shows error in proof of 2003 Ambainis distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs: Simulation shows that 2003 Childs–Eisenberg distinctness algorithm is non-functional; need to take half angle.

Childs: Yes. Typo, already fixed in 2005 journal version.

Do we know the best attacks?

Maybe, maybe not.

How many researchers have looked for better attacks?

Do those researchers have the right experience?

Simulations

2014.04 Chou → Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs:
Simulation shows that 2003
Childs–Eisenberg distinctness
algorithm is non-functional;
need to take half angle.

Childs: Yes. Typo, already
fixed in 2005 journal version.

Do we know the best attacks?

Maybe, maybe not.

How many researchers have
looked for better attacks?

Do those researchers
have the right experience?

Did they carefully study
all possible avenues of attack?

Simulations

2014.04 Chou → Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs:
Simulation shows that 2003
Childs–Eisenberg distinctness
algorithm is non-functional;
need to take half angle.

Childs: Yes. Typo, already
fixed in 2005 journal version.

Do we know the best attacks?

Maybe, maybe not.

How many researchers have
looked for better attacks?

Do those researchers
have the right experience?

Did they carefully study
all possible avenues of attack?

Is this auditable and audited?

Simulations

2014.04 Chou → Ambainis:
Simulation shows error in
proof of 2003 Ambainis
distinctness algorithm.

Ambainis: Yes, thanks, will fix.

2014.04 Chou → Childs:
Simulation shows that 2003
Childs–Eisenberg distinctness
algorithm is non-functional;
need to take half angle.

Childs: Yes. Typo, already
fixed in 2005 journal version.

Do we know the best attacks?

Maybe, maybe not.

How many researchers have
looked for better attacks?

Do those researchers
have the right experience?

Did they carefully study
all possible avenues of attack?

Is this auditable and audited?

Real-world security systems
cannot avoid these questions.