Quantum algorithms

Daniel J. Bernstein

"Quantum algorithm"
means an algorithm that
a quantum computer can run.

i.e. a sequence of instructions,
where each instruction is
in a quantum computer's
supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Quantum algorithms

Daniel J. Bernstein

"Quantum algorithm" means an algorithm that a quantum computer can run.

i.e. a sequence of instructions, where each instruction is in a quantum computer's supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1): contains many "qubits"; can efficiently perform "NOT gate", "Hadamard gate", "controlled NOT gate", "*T* gate".

Making these instructions work is the main goal of quantum-computer engineering.

Quantum algorithms

Daniel J. Bernstein

"Quantum algorithm"
means an algorithm that
a quantum computer can run.

i.e. a sequence of instructions,
where each instruction is
in a quantum computer's
supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
. . . "Simon's algorithm";
. . . "Shor's algorithm"; etc.

Quantum algorithms

Daniel J. Bernstein

"Quantum algorithm"
means an algorithm that
a quantum computer can run.

i.e. a sequence of instructions,
where each instruction is
in a quantum computer's
supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

m algorithms

. Bernstein

---

um algorithm"
n algorithm that
um computer can run.

quence of instructions,
ach instruction is
ntum computer's
ed instruction set.

we know which
ions a quantum
er will support?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "*T* gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
. . . "Simon's algorithm";
. . . "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantur
stores a
efficientl
laws of c
with as r

This is t
quantum
by 1982
physics v

ms

n

hm"
m that
ter can run.

instructions,
ction is
puter's
ion set.

which
antum
pport?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "*T* gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
. . . "Simon's algorithm";
. . . "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantum 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

n.

ns,

Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute “Toffoli gate”;
. . . “Simon’s algorithm”;
. . . “Shor’s algorithm”; etc.

General belief: Traditional CPU
isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (
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):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute “Toffoli gate”;
. . . “Simon’s algorithm”;
. . . “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman "Simulating
physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
"Quantum algorithms for
quantum field theories".

m computer type 1 (QC1):
many "qubits";
iently perform
ate", "Hadamard gate",
led NOT gate", "T gate".

these instructions work
nain goal of quantum-
er engineering.

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

belief: Traditional CPU
1; e.g. can't factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantur
efficientl
that any
compute

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

structions work
of quantum-
ering.

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

aditional CPU
n't factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum compute
efficiently compute
that any possible
computer can com

(QC1):

ate”,

gate”.

work

um-

CPU

quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman “Simulating
physics with computers”.

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
“Quantum algorithms for
quantum field theories”.

Quantum computer type 3 (
efficiently computes anythin
that any possible physical
computer can compute effici

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman "Simulating
physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
"Quantum algorithms for
quantum field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

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 possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note c

Apparen
Current
from D-
can be r
simulate

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

l concept of
rs introduced
"Simulating
uters".

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

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific
Current "quantum
from D-Wave are
can be more cost-
simulated by tradi

(QC2):

esired.

of
ced
ng

QC2.

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensu
Current "quantum compute
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CP

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

m computer type 3 (QC3):
y computes anything
possible physical
er can compute 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.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The stat

Data ("s
a list of
e.g.: (0,

er type 3 (QC3):
es anything
physical
npute efficiently.

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

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

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a cor

Data ("state") sto
a list of 3 element
e.g.: $(0, 0, 0)$.

(QC3):

g

iently.

n QC3.

ently

uter.

n QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 b
a list of 3 elements of $\{0, 1\}$
e.g.: $(0, 0, 0)$.

A note on D-Wave

Apparent scientific consensus: Current “quantum computers” from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data (“state”) stored in 3 bits: a list of 3 elements of $\{0, 1\}$. e.g.: $(0, 0, 0)$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits: a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits: a list of 64 elements of $\{0, 1\}$.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits: a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits: a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

on D-Wave

t scientific consensus:
"quantum computers"
Wave are useless—
more cost-effectively
d by traditional CPUs.

Vave is
ting venture capital;
some machines;
ting possibly useful
eering expertise;
eing punished
ceiving people.

ve a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The stat

Data sto
a list of
e.g.: (3,

e

c consensus:
n computers”
useless—
effectively
tional CPUs.

re capital;
chines;
bly useful
ertise;
ned
ople.

nvestment?

The state of a computer

Data (“state”) stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a qua

Data stored in 3 q
a list of 8 numbers
e.g.: $(3, 1, 4, 1, 5, 9$

s:

rs"

Us.

t?

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum com

Data stored in 3 qubits:

a list of 8 numbers, not all z

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:

$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:

a list of 2^{64} numbers, not all zero.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.
e.g.: $(0,1,1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0,1\}$.
e.g.: $(1,1,1,1,1,0,0,0,1,$
$0,0,0,0,0,0,1,1,0,0,0,$
$0,1,0,0,1,0,0,0,0,0,1,$
$1,0,1,0,0,0,1,0,0,0,1,$
$0,0,1,1,1,0,0,1,0,0,0,$
$1,1,0,1,1,0,0,1,0,0,1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3,1,4,1,5,9,2,6)$.
e.g.: $(-2,7,-1,8,1,-8,-2,8)$.
e.g.: $(0,0,0,0,0,1,0,0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

te of a computer

state") stored in 3 bits:

3 elements of $\{0, 1\}$.

$0, 0)$.

$1, 1)$.

$1, 1)$.

ored in 64 bits:

64 elements of $\{0, 1\}$.

$1, 1, 1, 1, 0, 0, 0, 1,$

$, 0, 0, 1, 1, 0, 0, 0,$

$, 1, 0, 0, 0, 0, 0, 1,$

$, 0, 0, 1, 0, 0, 0, 1,$

$, 1, 0, 0, 1, 0, 0, 0,$

$, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuri

Can sim

Cannot s

of numb

mputer

red in 3 bits:

s of $\{0, 1\}$.

bits:

ts of $\{0, 1\}$.

$0, 0, 0, 1,$

$0, 0, 0,$

$0, 0, 1,$

$0, 0, 1,$

$0, 0, 0,$

$0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: $(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quant

Can simply look a

Cannot simply loo

of numbers stored

its:

.

}.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: $(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum comp

Can simply look at a bit.
Cannot simply look at the li
of numbers stored in n qubi

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

te of a quantum computer

ored in 3 qubits:

8 numbers, not all zero.

$1, 4, 1, 5, 9, 2, 6)$.

$2, 7, -1, 8, 1, -8, -2, 8)$.

$0, 0, 0, 0, 1, 0, 0)$.

ored in 4 qubits: a list of

bers, not all zero. e.g.:

$1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

ored in 64 qubits:

2^{64} numbers, not all zero.

ored in 1000 qubits: a list

numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list
of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measurement produces q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros
except 1 at position q.

e.g.: Say

$(1, 1, 1, 1$

antum computer

ubits:

s, not all zero.

9, 2, 6).

, 1, −8, −2, 8).

1, 0, 0).

ubits: a list of

ll zero. e.g.:

, 5, 3, 5, 8, 9, 7, 9, 3).

qubits:

ers, not all zero.

00 qubits: a list

not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list
of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measurement produces q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros
except 1 at position q.

e.g.: Say 3 qubits

(1, 1, 1, 1, 1, 1, 1, 1

nputer

zero.

2, 8).

st of

g.:

9, 7, 9, 3).

l zero.

a list

ro.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have stat
$(1, 1, 1, 1, 1, 1, 1, 1)$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits
- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

ng a quantum computer

ply look at a bit.
simply look at the list
ers stored in n qubits.

ing n qubits
ces n bits and
ys the state.

its have state
$\ldots, a_{2^n-1})$ then
ment produces q
bability $|a_q|^2 / \sum_r |a_r|^2$.

then all zeros
at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say
$(3, 1, 4,$

tum computer

t a bit.

k at the list

in n qubits.

its

and

te.

ate

then

luces q

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

eros

on q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces

$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits

$(3, 1, 4, 1, 5, 9, 2, 6$

uter

st

ts.

$a_r|^2$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

y 3 qubits have state
$1, 1, 1, 1, 1)$.

ement produces
with probability $1/8$;
with probability $1/8$;
with probability $1/8$;
with probability $1/8$;
with probability $1/8$;
with probability $1/8$;
with probability $1/8$;
with probability $1/8$.

um RNG.”

g: Quantum RNGs sold
e measurably biased.

e.g.: Say 3 qubits have state
$(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say
$(0, 0, 0, 0$

have state

).

duces

ability $1/8$;

ability $1/8$;

ability $1/8$;

ability $1/8$;

ability $1/8$;

ability $1/8$;

ability $1/8$;

ability $1/8$.

m RNGs sold

bly biased.

e.g.: Say 3 qubits have state

$(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces

$000 = 0$ with probability $9/173$;

$001 = 1$ with probability $1/173$;

$010 = 2$ with probability $16/173$;

$011 = 3$ with probability $1/173$;

$100 = 4$ with probability $25/173$;

$101 = 5$ with probability $81/173$;

$110 = 6$ with probability $4/173$;

$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits

$(0, 0, 0, 0, 0, 1, 0, 0$

e

3;

3;

3;

3;

3;

3;

3;

3.

old

.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces

$000 = 0$ with probability $9/173$;

$001 = 1$ with probability $1/173$;

$010 = 2$ with probability $16/173$;

$011 = 3$ with probability $1/173$;

$100 = 4$ with probability $25/173$;

$101 = 5$ with probability $81/173$;

$110 = 6$ with probability $4/173$;

$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

e.g.: Say 3 qubits have state
$(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state
$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

y 3 qubits have state
$1, 5, 9, 2, 6)$.

ement produces
with probability $9/173$;
with probability $1/173$;
with probability $16/173$;
with probability $1/173$;
with probability $25/173$;
with probability $81/173$;
with probability $4/173$;
with probability $36/173$.

t likely outcome.

e.g.: Say 3 qubits have state
$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT ga

NOT_0 g
$(3, 1, 4, 1$
$(1, 3, 1, 4$

have state

).

duces

ability $9/173$;

ability $1/173$;

ability $16/173$;

ability $1/173$;

ability $25/173$;

ability $81/173$;

ability $4/173$;

ability $36/173$.

tcome.

e.g.: Say 3 qubits have state

$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

$000 = 0$ with probability 0;

$001 = 1$ with probability 0;

$010 = 2$ with probability 0;

$011 = 3$ with probability 0;

$100 = 4$ with probability 0;

$101 = 5$ with probability 1;

$110 = 6$ with probability 0;

$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 q

$(3, 1, 4, 1, 5, 9, 2, 6$

$(1, 3, 1, 4, 9, 5, 6, 2$

e

173;

173;

/173;

173;

/173;

/173;

173;

/173.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

$000 = 0$ with probability 0;

$001 = 1$ with probability 0;

$010 = 2$ with probability 0;

$011 = 3$ with probability 0;

$100 = 4$ with probability 0;

$101 = 5$ with probability 1;

$110 = 6$ with probability 0;

$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.

e.g.: Say 3 qubits have state $(0,0,0,0,0,1,0,0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3,1,4,1,5,9,2,6) \mapsto (1,3,1,4,9,5,6,2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto (1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

e.g.: Say 3 qubits have state
$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

<u>NOT gates</u>

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

e.g.: Say 3 qubits have state
$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

y 3 qubits have state
0, 0, 1, 0, 0).

ement produces
with probability 0;
with probability 0;
with probability 0;
with probability 0;
with probability 0;
with probability 1;
with probability 0;
with probability 0.

ranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

(1, 0, 0,
(0, 1, 0,
(0, 0, 1,
(0, 0, 0,
(0, 0, 0,
(0, 0, 0,
(0, 0, 0,
(0, 0, 0,

Operatio
NOT_0, s
Operatio
flipping
Flip: ou

have state

).

duces

ability 0;

ability 0;

ability 0;

ability 0;

ability 0;

ability 1;

ability 0;

ability 0.

tcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

state

(1, 0, 0, 0, 0, 0, 0, 0

(0, 1, 0, 0, 0, 0, 0, 0

(0, 0, 1, 0, 0, 0, 0, 0

(0, 0, 0, 1, 0, 0, 0, 0

(0, 0, 0, 0, 1, 0, 0, 0

(0, 0, 0, 0, 0, 1, 0, 0

(0, 0, 0, 0, 0, 0, 1, 0

(0, 0, 0, 0, 0, 0, 0, 1

Operation on quan
NOT_0, swapping p
Operation after m
flipping bit 0 of re
Flip: output is not

e

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

state	measure
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state
NOT_0, swapping pairs.
Operation after measuremen
flipping bit 0 of result.
Flip: output is not input.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(1, 3, 1, 4, 9, 5, 6, 2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(4, 1, 3, 1, 2, 6, 5, 9)$.

NOT_2 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(5, 9, 2, 6, 3, 1, 4, 1)$.

state	measurement
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

tes

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$4, 9, 5, 6, 2)$.

ate on 4 qubits:

$5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$

$9,5,6,2,3,5,8,5,7,9,3,9)$.

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$1, 2, 6, 5, 9)$.

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$6, 3, 1, 4, 1)$.

state	measurement
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controll

e.g. $\mathrm{C}_1\mathrm{N}$

$(3, 1, 4,$

$(3, 1, 1,$

ubits:

) $\mapsto$

).

ubits:

3,5,8,9,7,9,3) $\mapsto$

5,8,5,7,9,3,9).

ubits:

) $\mapsto$

).

ubits:

) $\mapsto$

).

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (C

e.g. C_1NOT_0:
$(3,1,4,1,5,9,2,6$
$(3,1,1,4,5,9,6,2$

$,3) \mapsto$
$,9).$

state	measurement
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state: NOT_0, swapping pairs.

Operation after measurement: flipping bit 0 of result.

Flip: output is not input.

<u>Controlled-NOT (CNOT) ga</u>

e.g. $\mathrm{C}_1\mathrm{NOT}_0$:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2).$

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (CNOT) gates

e.g. $\mathrm{C}_1\mathrm{NOT}_0$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

state	measurement
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (CNOT) gates

e.g. $\mathrm{C}_1\mathrm{NOT}_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\mathrm{C}_2\mathrm{NOT}_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,9,5,6,2)$.

e.g. C_0NOT_2:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,9,4,6,5,1,2,1)$.

state	measurement
$0,0,0,0,0)$	000
$0,0,0,0,0)$	001
$0,0,0,0,0)$	010
$1,0,0,0,0)$	011
$0,1,0,0,0)$	100
$0,0,1,0,0)$	101
$0,0,0,1,0)$	110
$0,0,0,0,1)$	111

on on quantum state:
swapping pairs.
on after measurement:
bit 0 of result.
tput is not input.

Controlled-NOT (CNOT) gates

e.g. $\mathrm{C}_1\mathrm{NOT}_0$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\mathrm{C}_2\mathrm{NOT}_0$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,9,5,6,2)$.

e.g. $\mathrm{C}_0\mathrm{NOT}_2$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,9,4,6,5,1,2,1)$.

Toffoli g

Also kno
controlle

e.g. $\mathrm{C}_2\mathrm{C}$
$(3,1,4,1$
$(3,1,4,1$

measurement

0) 000

0) 001

0) 010

0) 011

0) 100

0) 101

0) 110

1) 111

ntum state:

pairs.

easurement:

sult.

t input.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$

$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:

flipping bit 0 *if* bit 1 is set; i.e.,

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

e.g. C_2NOT_0:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$

$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. C_0NOT_2:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$

$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as CC

controlled-controll

e.g. $C_2C_1NOT_0$:

$(3, 1, 4, 1, 5, 9, 2, 6$

$(3, 1, 4, 1, 5, 9, 6, 2$

ement

e:

nt:

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. C_0NOT_2:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as CCNOT gate
controlled-controlled-NOT g

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,9,5,6,2)$.

e.g. C_0NOT_2:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,9,4,6,5,1,2,1)$.

Toffoli gates

Also known as CCNOT gates:
controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,5,9,6,2)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. C_0NOT_2:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as CCNOT gates:
controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

Controlled-NOT (CNOT) gates

e.g. C_1NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. C_2NOT_0:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. C_0NOT_2:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as CCNOT gates:
controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $C_0C_1NOT_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

ed-NOT (CNOT) gates

IOT$_0$:

1, 5, 9, 2, 6) $\mapsto$

4, 5, 9, 6, 2).

on after measurement:

bit 0 *if* bit 1 is set; i.e.,

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

IOT$_0$:

1, 5, 9, 2, 6) $\mapsto$

1, 9, 5, 6, 2).

IOT$_2$:

1, 5, 9, 2, 6) $\mapsto$

6, 5, 1, 2, 1).

Toffoli gates

Also known as CCNOT gates:
controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $C_0C_1NOT_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More sh

Combine

to build

CNOT) gates

) $\mapsto$

).

easurement:

t 1 is set; i.e.,

$, q_1, q_0 \oplus q_1)$.

) $\mapsto$

).

) $\mapsto$

).

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $\mathrm{C}_2\mathrm{C}_1\mathrm{NOT}_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\mathrm{C}_0\mathrm{C}_1\mathrm{NOT}_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, C
to build other per

ates

nt:

i.e.,

q_1).

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $\mathrm{C_2C_1NOT_0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\mathrm{C_0C_1NOT_2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toff
to build other permutations.

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $C_0C_1NOT_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

Toffoli gates

Also known as CCNOT gates: controlled-controlled-NOT gates.

e.g. $C_2C_1NOT_0$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

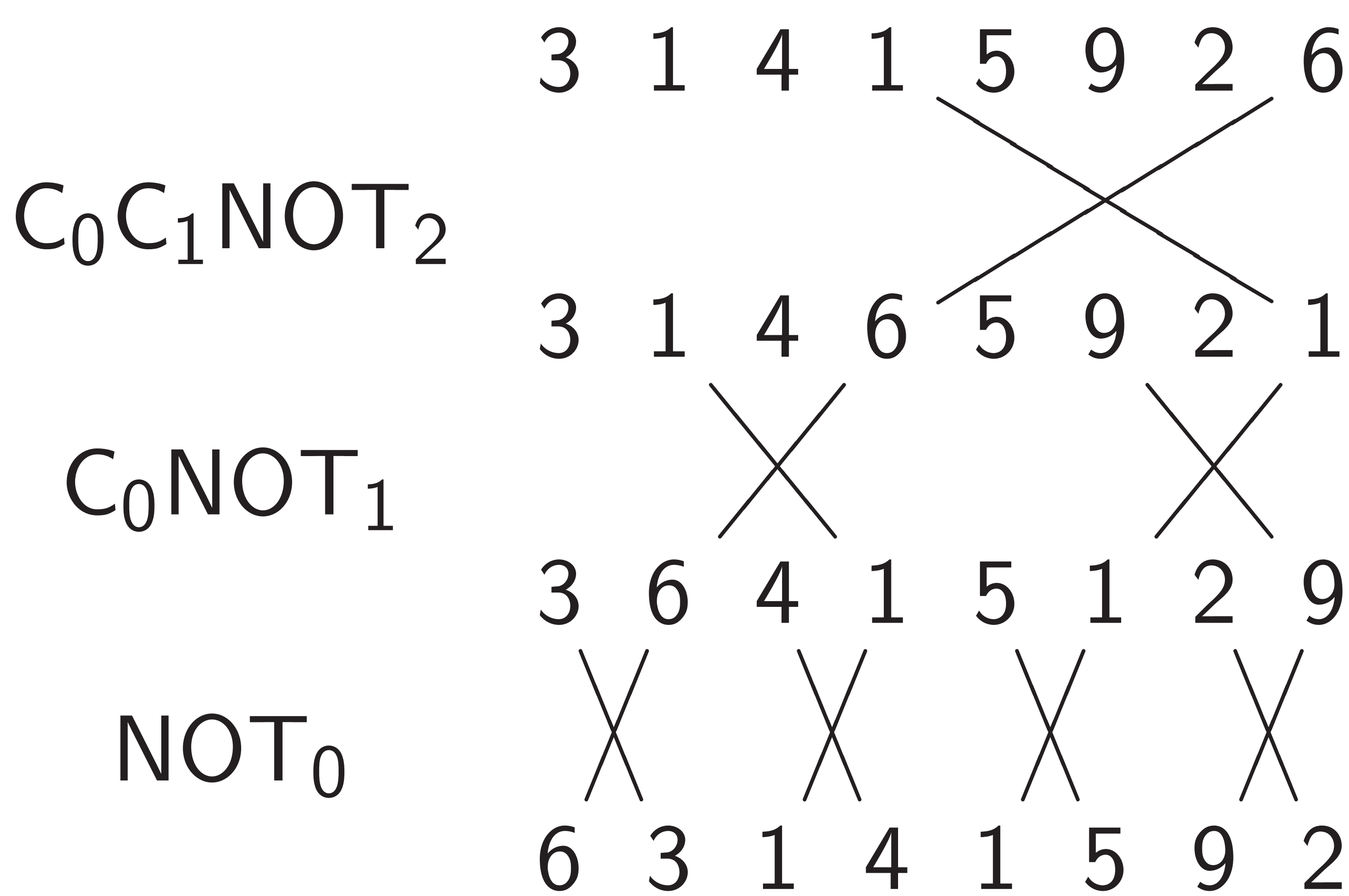
Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $C_0C_1NOT_2$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:



gates

own as CCNOT gates:
ed-controlled-NOT gates.

C_1NOT_0:
$1, 5, 9, 2, 6) \mapsto$
$1, 5, 9, 6, 2)$.

on after measurement:
$q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

C_1NOT_2:
$1, 5, 9, 2, 6) \mapsto$
$6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

e.g. series of gates to
rotate 8 positions by distance 1:

	3	1	4	1	5	9	2	6
$C_0C_1NOT_2$								
	3	1	4	6	5	9	2	1
C_0NOT_1								
	3	6	4	1	5	1	2	9
NOT_0								
	6	3	1	4	1	5	9	2

Hadama

Hadama

$(a, b) \mapsto$

3 1
4 2

NOT gates:
ed-NOT gates.

$) \mapsto$
$).$

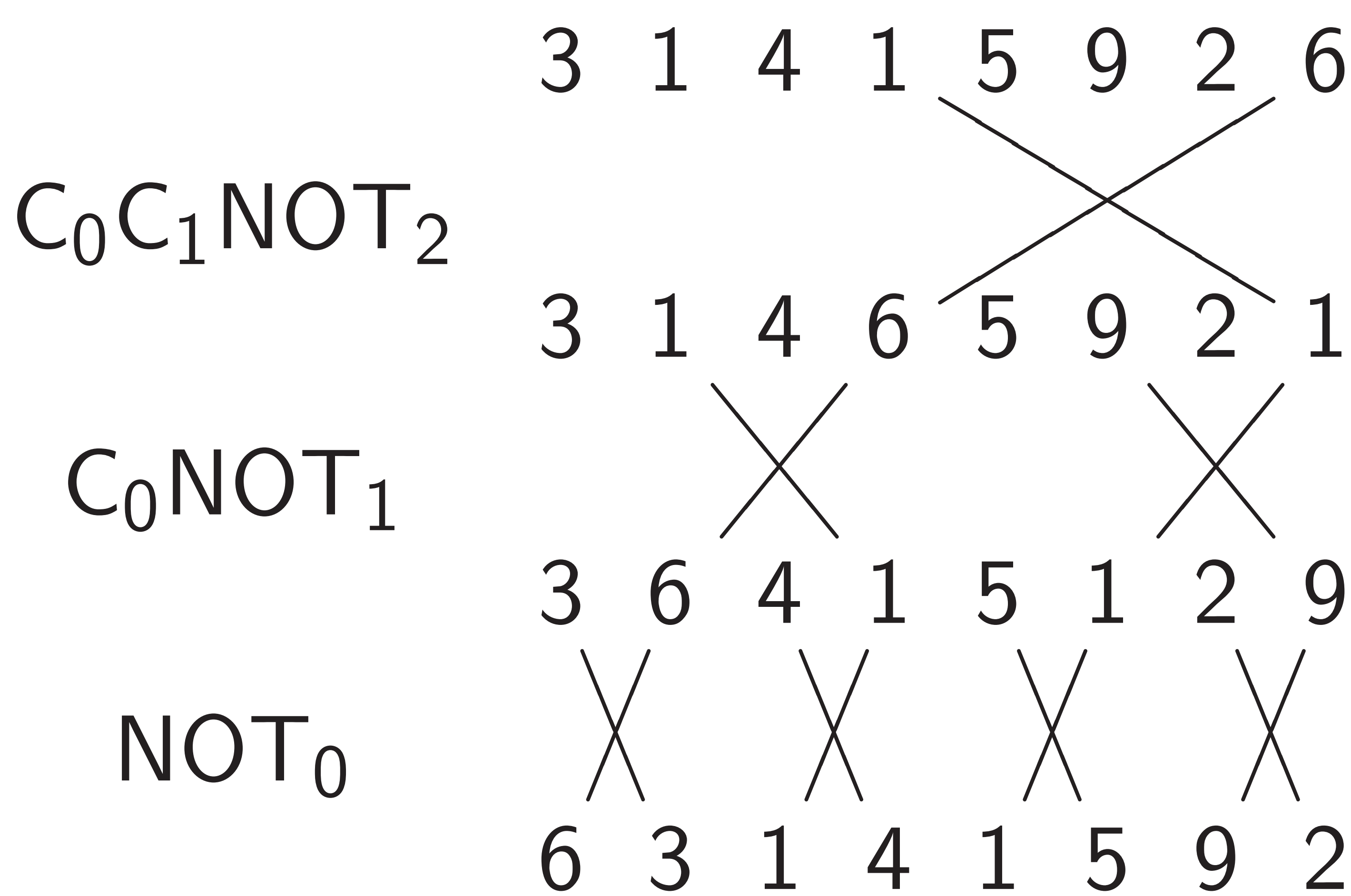
easurement:
$, q_1, q_0 \oplus q_1 q_2).$

$) \mapsto$
$).$

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

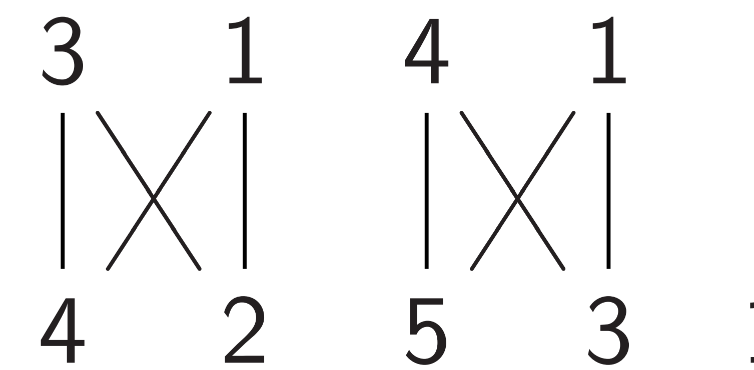
e.g. series of gates to
rotate 8 positions by distance 1:



Hadamard gates

Hadamard$_0$:

$(a, b) \mapsto (a + b, a$



es:

gates.

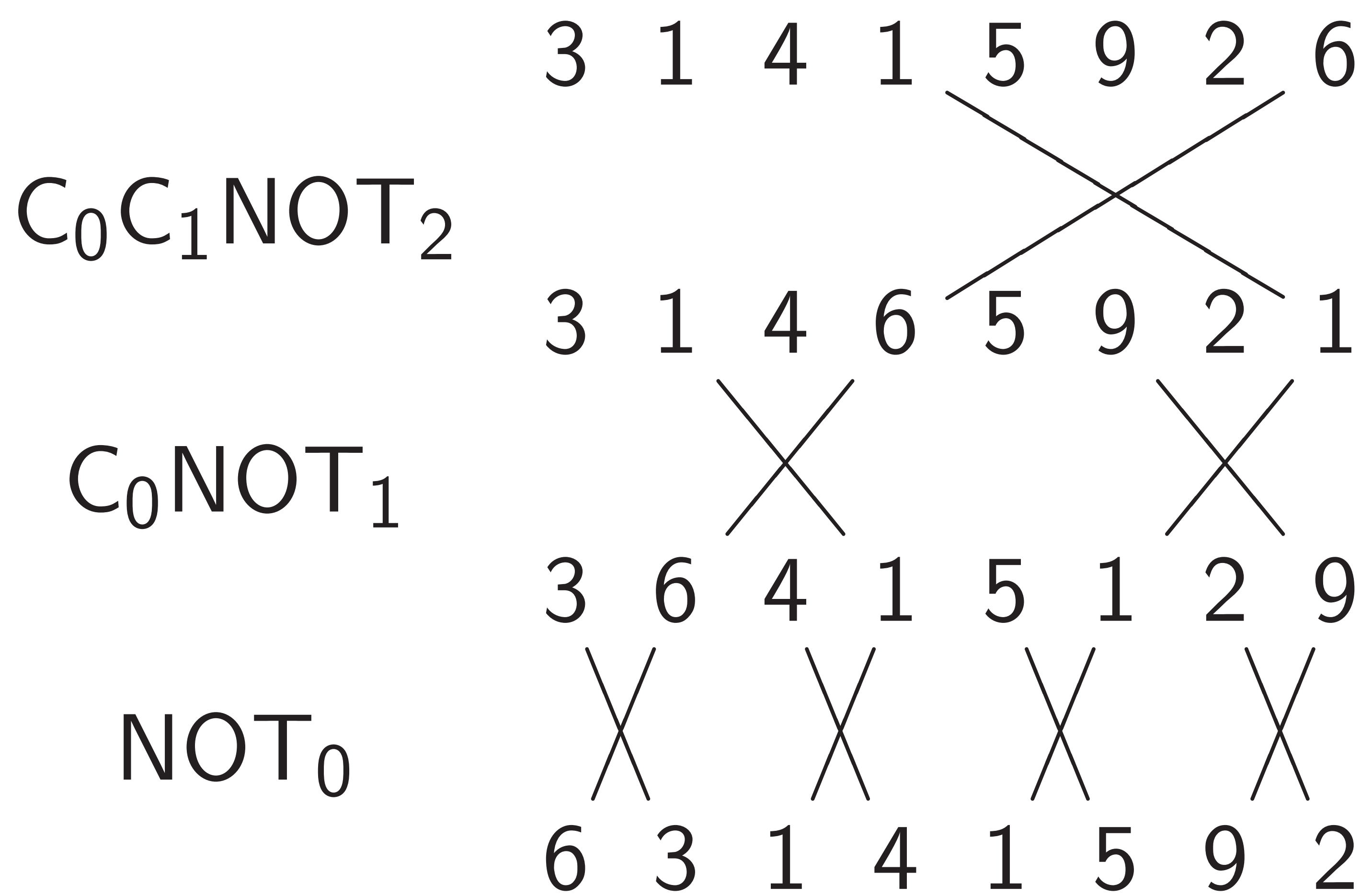
nt:

$q_1 q_2)$.

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

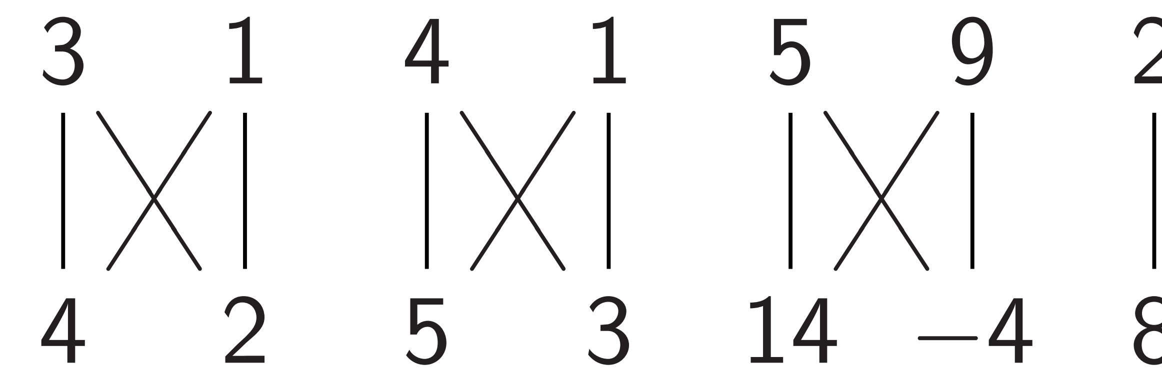
e.g. series of gates to rotate 8 positions by distance 1:



Hadamard gates

Hadamard_0:

$(a, b) \mapsto (a + b, a - b)$.



More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

e.g. series of gates to
rotate 8 positions by distance 1:

	3	1	4	1	5	9	2	6
$C_0C_1NOT_2$								
	3	1	4	6	5	9	2	1
C_0NOT_1								
	3	6	4	1	5	1	2	9
NOT_0								
	6	3	1	4	1	5	9	2

Hadamard gates

Hadamard_0:

$(a, b) \mapsto (a + b, a - b)$.

3	1	4	1	5	9	2	6
4	2	5	3	14	−4	8	−4

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:

3 1 4 1 5 9 2 6

$C_0C_1NOT_2$

3 1 4 6 5 9 2 1

C_0NOT_1

3 6 4 1 5 1 2 9

NOT_0

6 3 1 4 1 5 9 2

Hadamard gates

Hadamard_0:

$(a, b) \mapsto (a + b, a - b)$.

3 1 4 1 5 9 2 6

4 2 5 3 14 −4 8 −4

Hadamard_1:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.

3 1 4 1 5 9 2 6

7 2 −1 0 7 15 3 3

uffling

e NOT, CNOT, Toffoli
other permutations.

es of gates to
positions by distance 1:

3 1 4 1 5 9 2 6

OT$_2$

3 1 4 6 5 9 2 1

T$_1$

3 6 4 1 5 1 2 9

$_0$

6 3 1 4 1 5 9 2

Hadamard gates

Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.

3 1 4 1 5 9 2 6

4 2 5 3 14 −4 8 −4

Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.

3 1 4 1 5 9 2 6

7 2 −1 0 7 15 3 3

Some us

Hadama

3 1

4 2

2 4

6 −2

NOT, Toffoli
mutations.

s to
by distance 1:

4 1 5 9 2 6

4 6 5 9 2 1

4 1 5 1 2 9

1 4 1 5 9 2

Hadamard gates

Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.

3 1 4 1 5 9 2 6

4 2 5 3 14 -4 8 -4

Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.

3 1 4 1 5 9 2 6

7 2 -1 0 7 15 3 3

Some uses of Had

Hadamard$_0$, NOT

3 1 4 1

4 2 5 3

2 4 3 5

6 -2 8 -2

foli

ce 1:

9 2 6

9 2 1

1 2 9

5 9 2

Hadamard gates

Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.

3	1	4	1	5	9	2	6
4	2	5	3	14	−4	8	−4

Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.

3	1	4	1	5	9	2	6
7	2	−1	0	7	15	3	3

Some uses of Hadamard gat

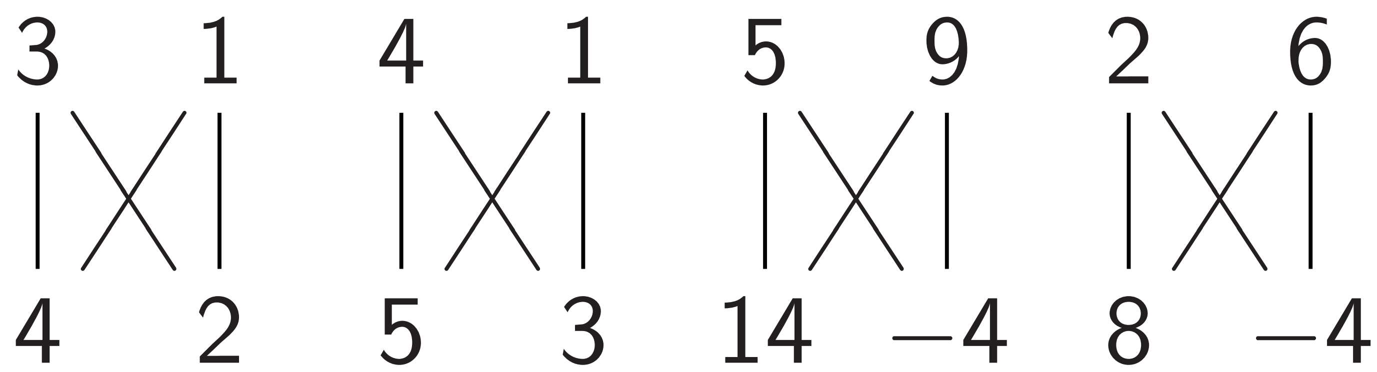
Hadamard$_0$, NOT$_0$, Hadama

3	1	4	1	5	9	2
4	2	5	3	14	−4	8
2	4	3	5	−4	14	−
6	−2	8	−2	10	−18	4

Hadamard gates

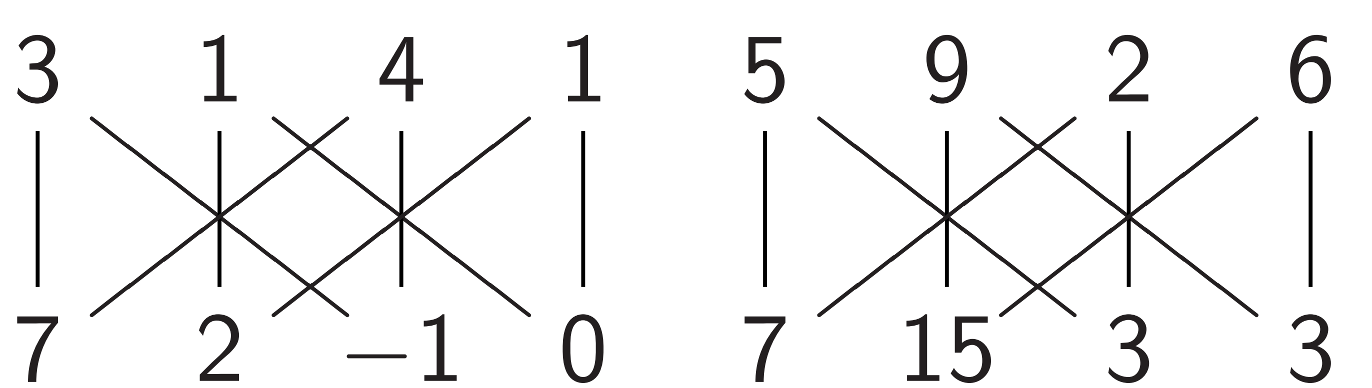
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



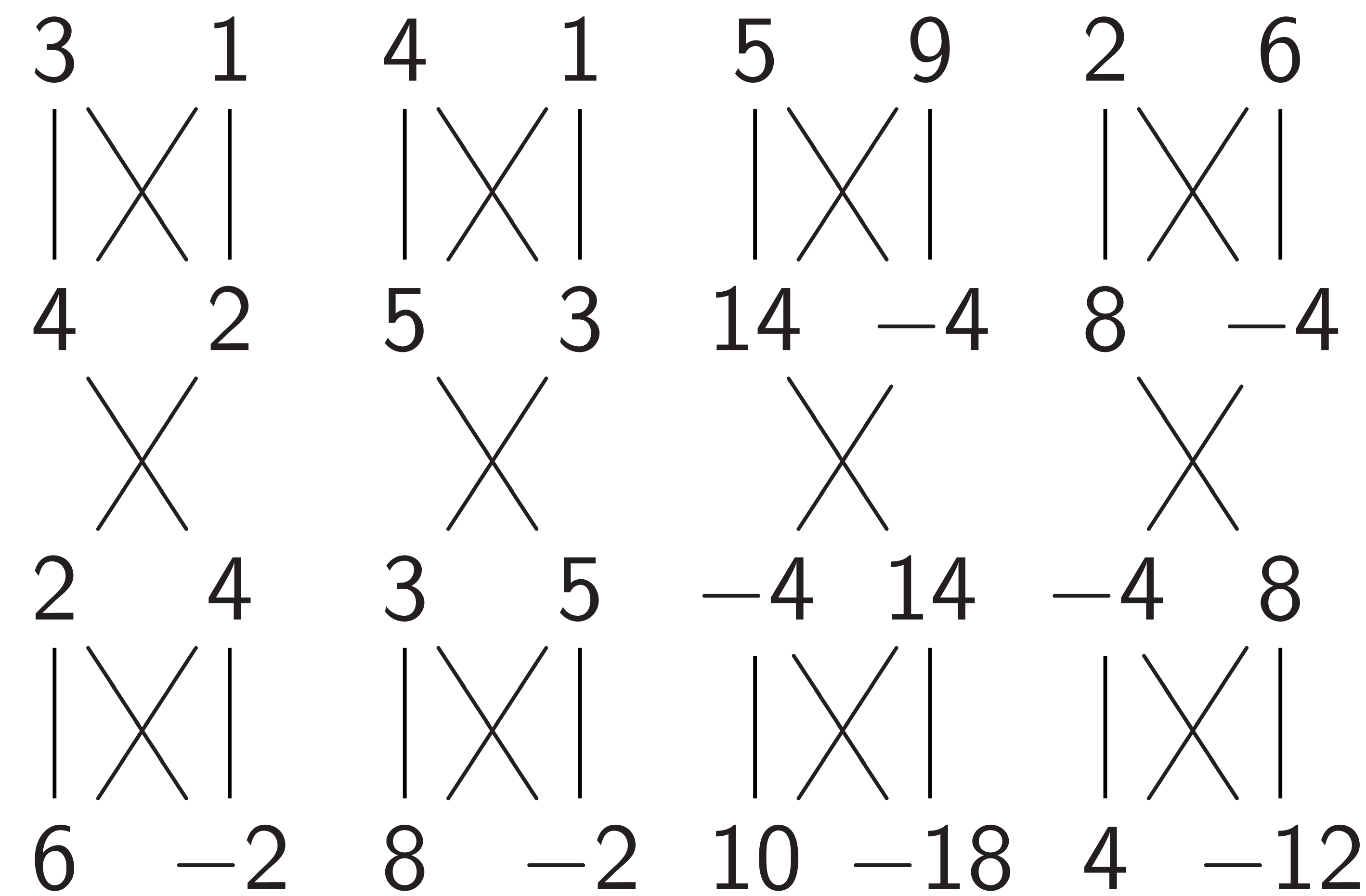
Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Some uses of Hadamard gates

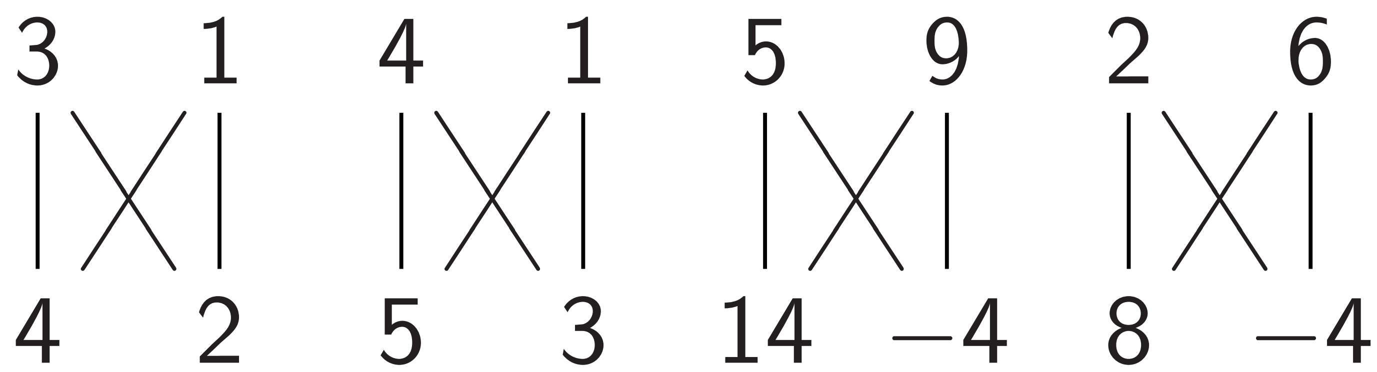
Hadamard$_0$, NOT$_0$, Hadamard$_0$:



Hadamard gates

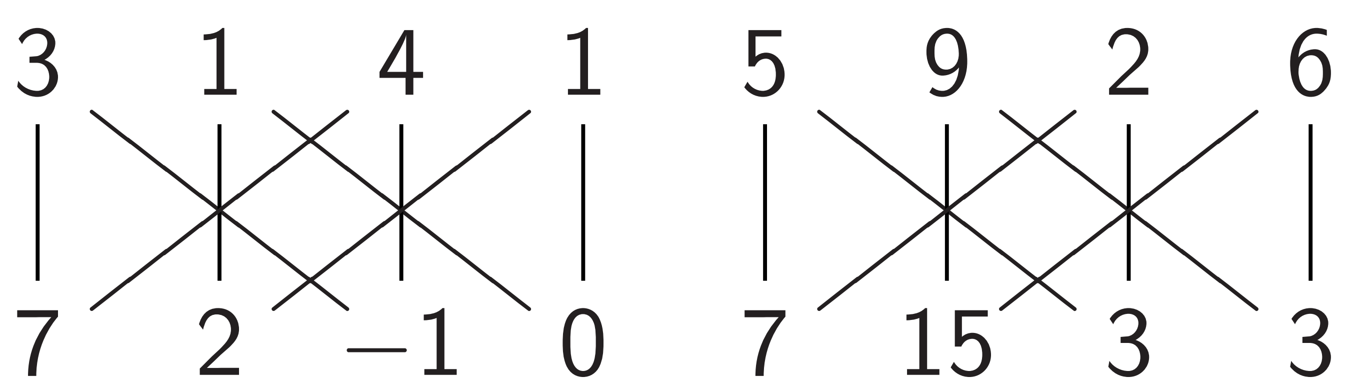
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



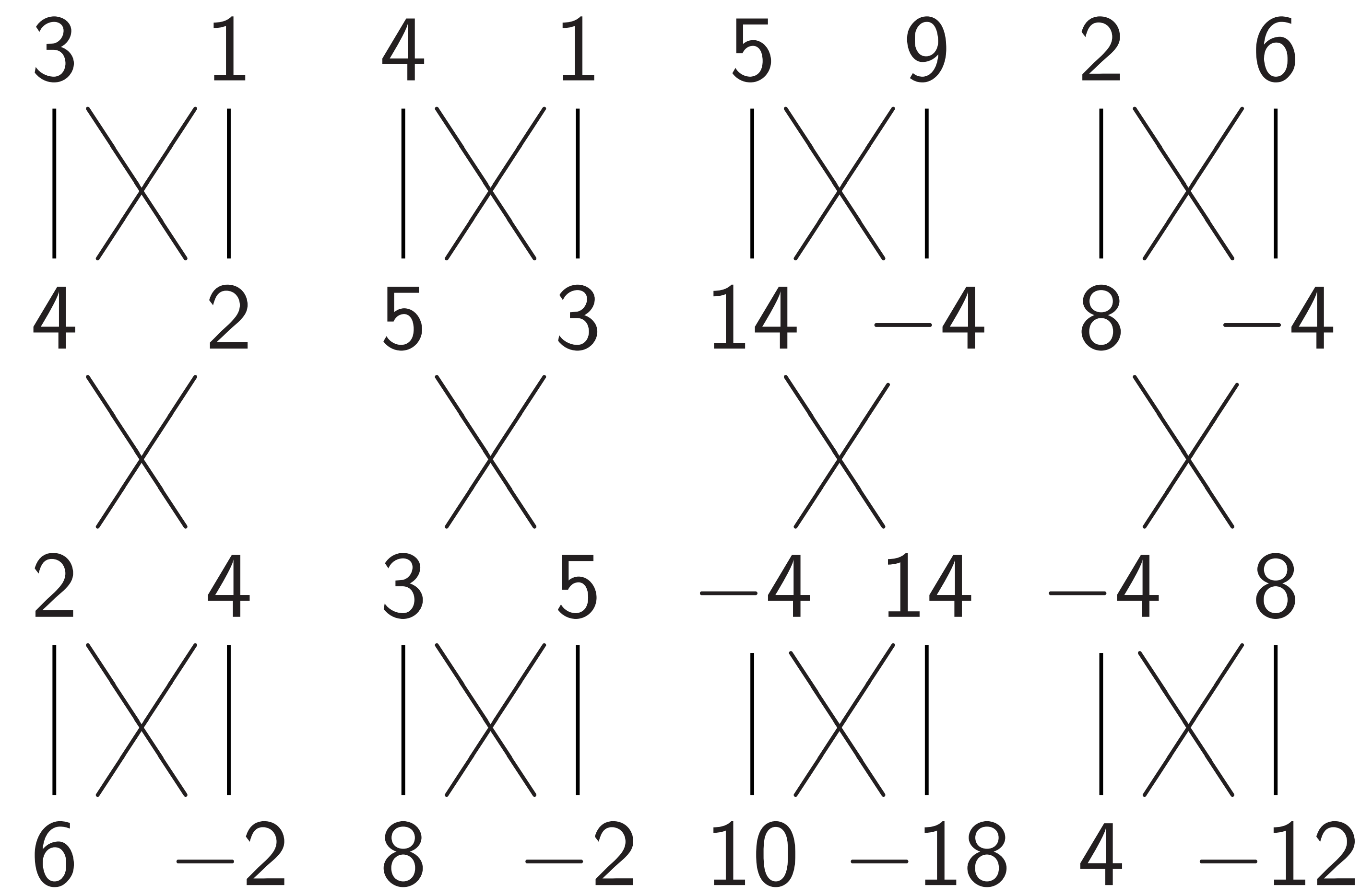
Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:

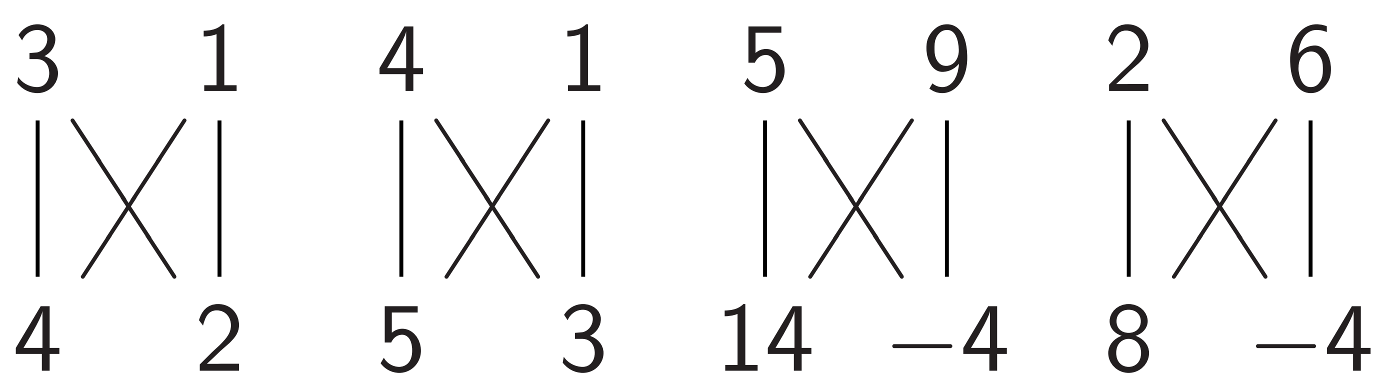


“Multiply each amplitude by 2.”
This is not physically observable.

Hadamard gates

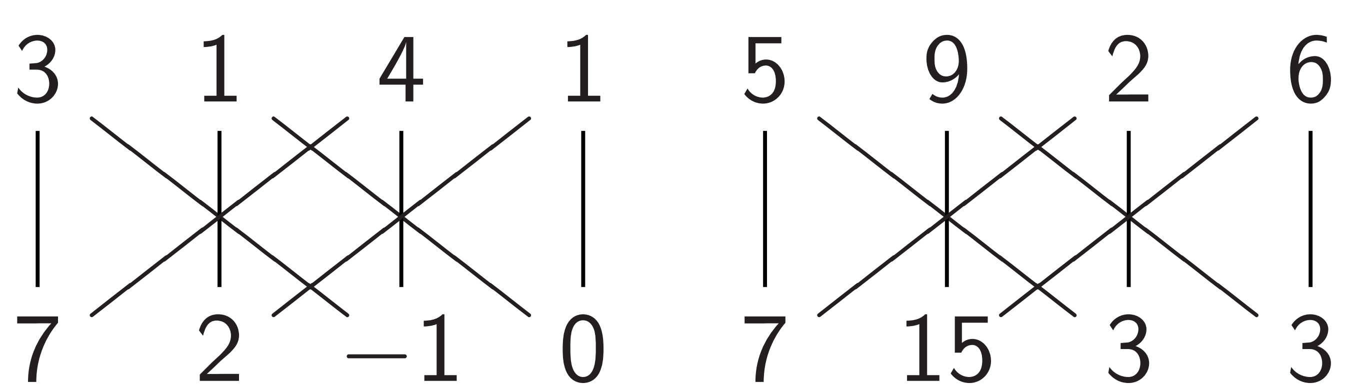
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



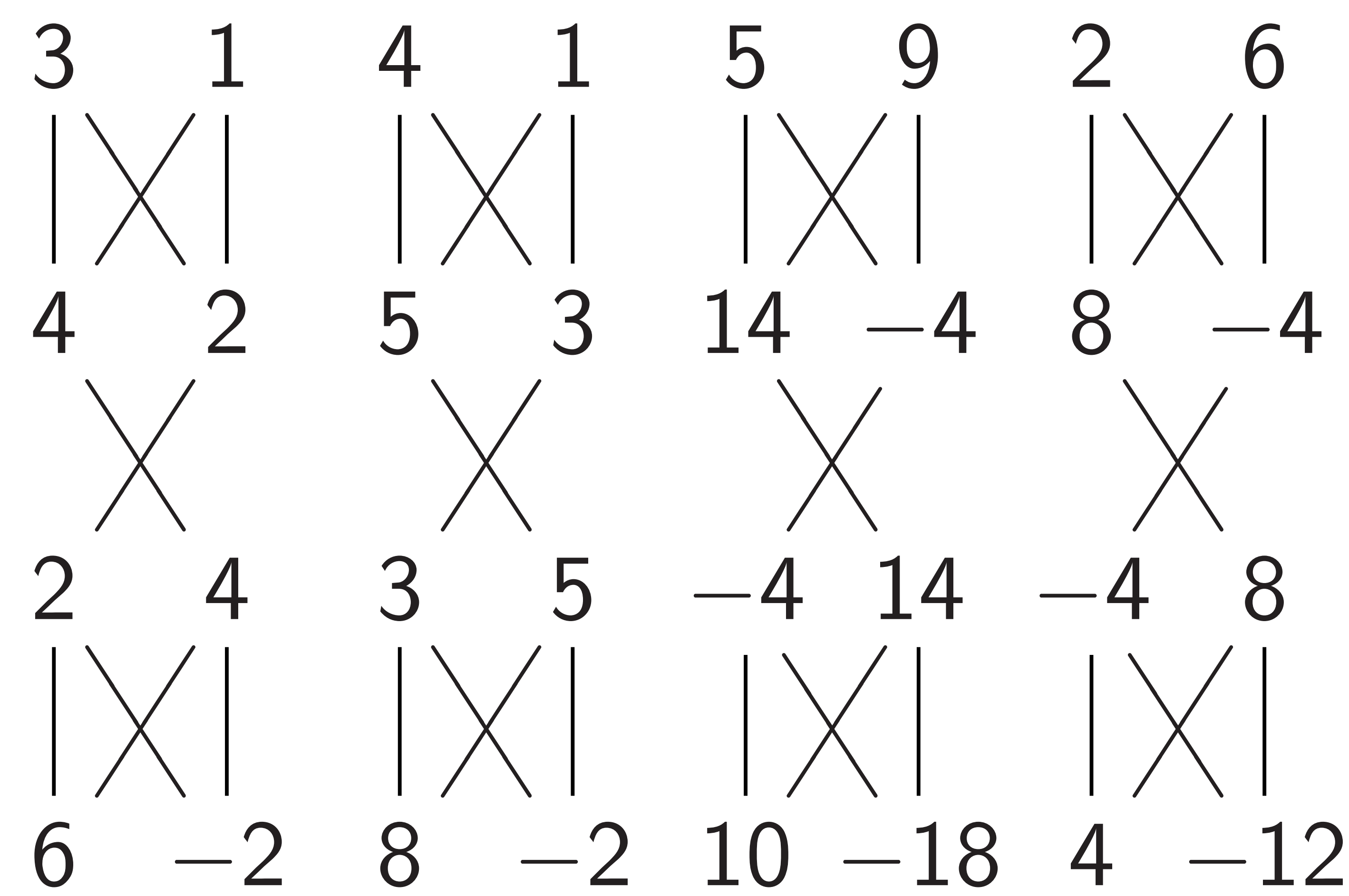
Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:



"Multiply each amplitude by 2."
This is not physically observable.

"Negate amplitude if q_0 is set."
No effect on measuring *now*.

rd gates

rd$_0$:

$\mapsto (a + b, a - b)$.

4 1 5 9 2 6

5 3 14 −4 8 −4

rd$_1$:

$d) \mapsto$

$b + d, a - c, b - d)$.

4 1 5 9 2 6

−1 0 7 15 3 3

Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:

3 1 4 1 5 9 2 6

4 2 5 3 14 −4 8 −4

2 4 3 5 −4 14 −4 8

6 −2 8 −2 10 −18 4 −12

"Multiply each amplitude by 2."
This is not physically observable.

"Negate amplitude if q_0 is set."
No effect on measuring *now*.

Fancier

"Negate

Assumes

C_0C_1NO

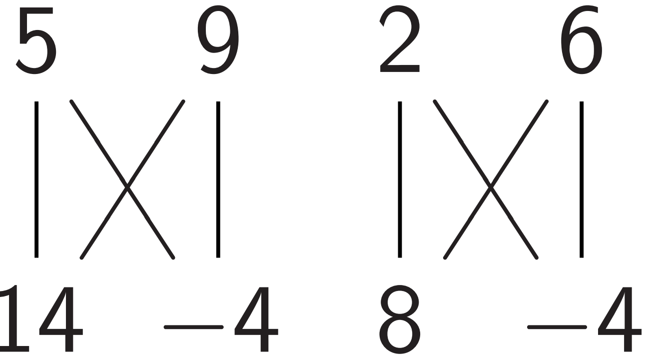
Hadama

NOT

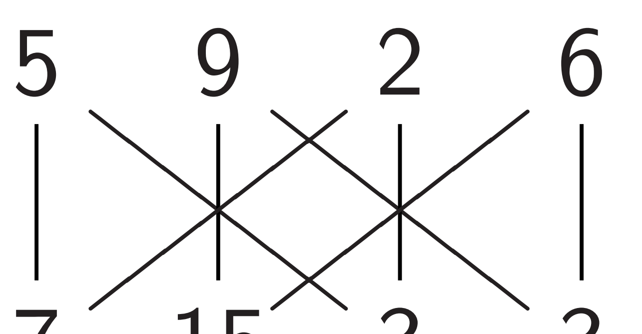
Hadama

C_0C_1NO

$- b)$.

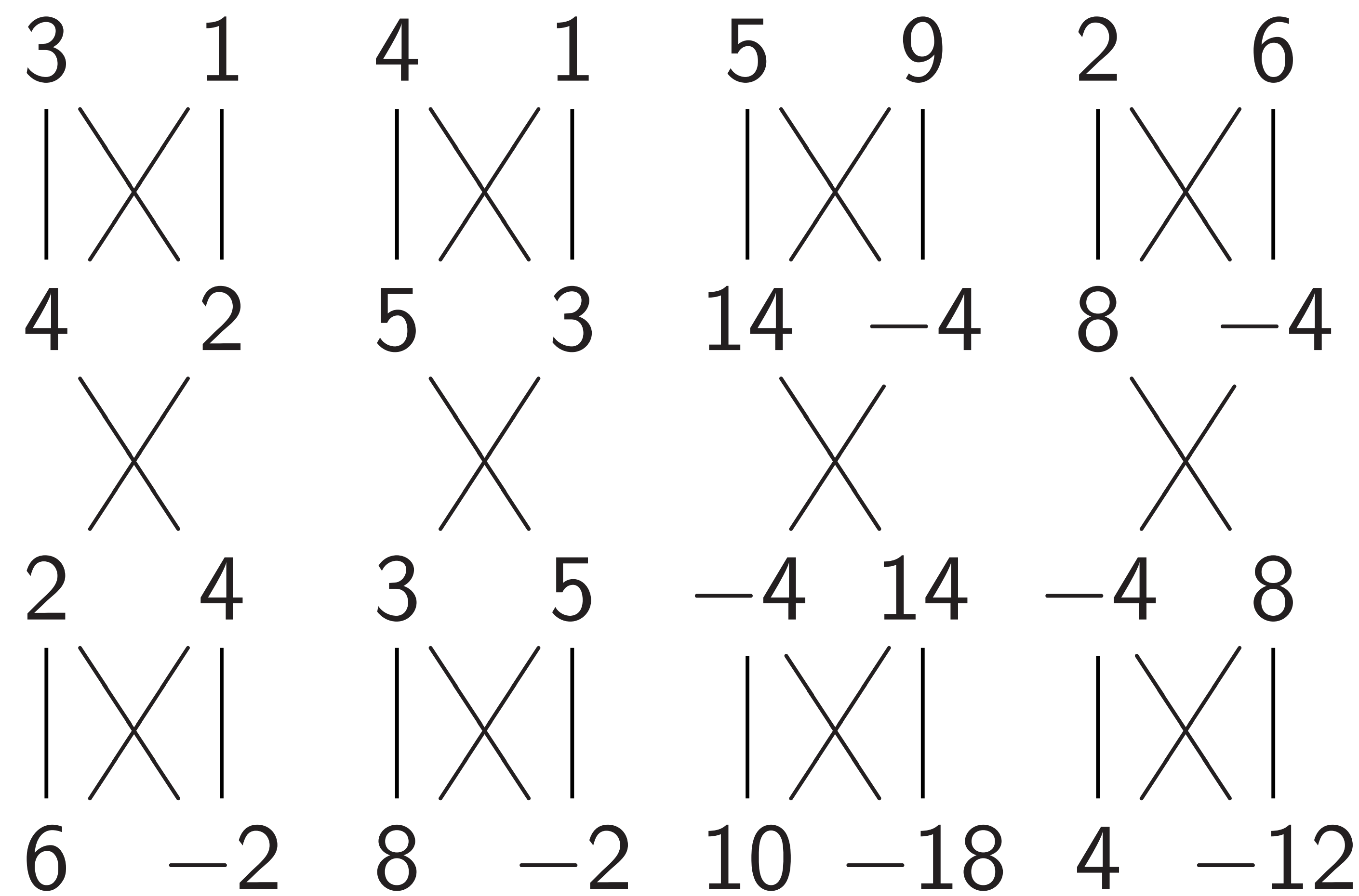


$c, b - d)$.



Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:



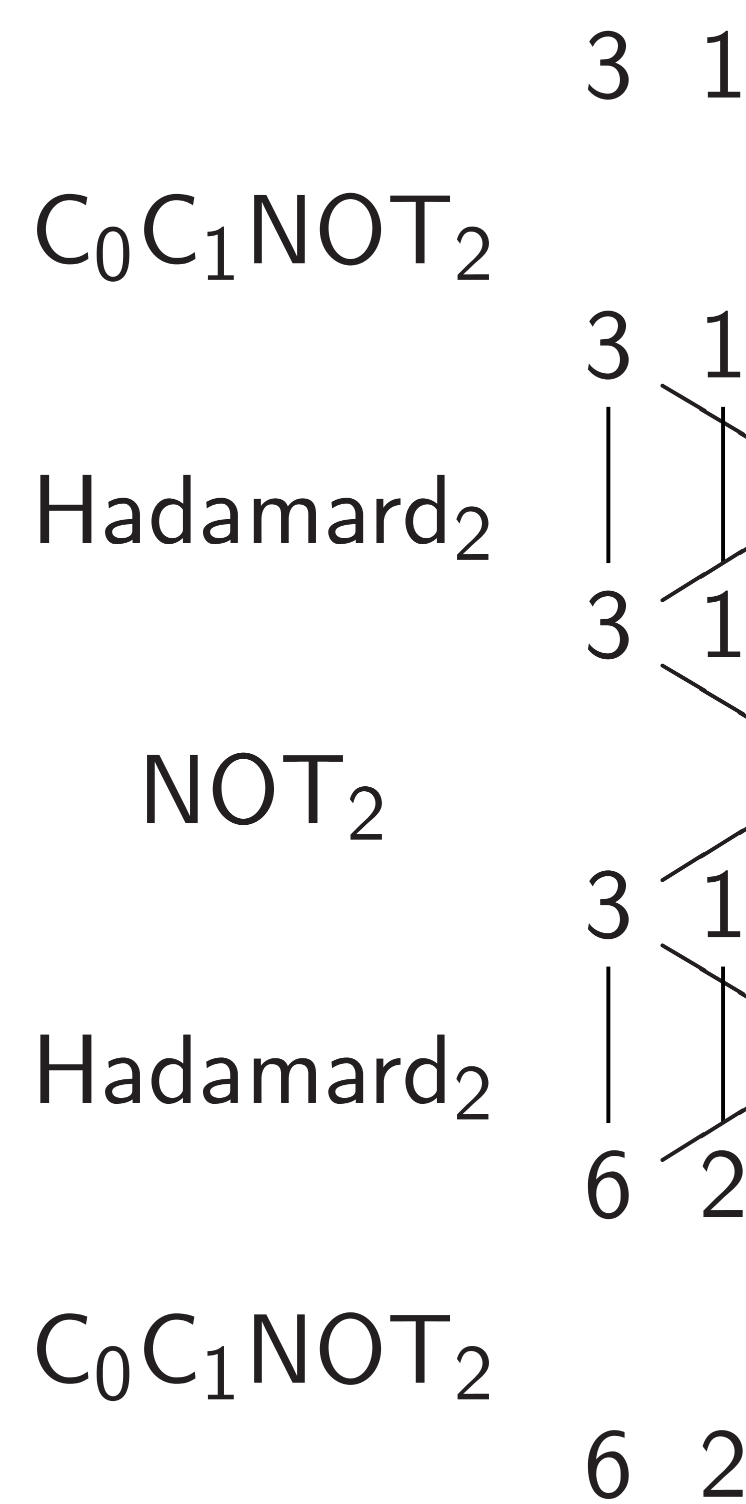
“Multiply each amplitude by 2.”
This is not physically observable.

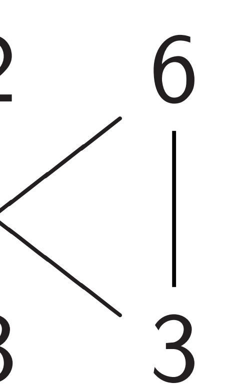
“Negate amplitude if q_0 is set.”
No effect on measuring *now*.

Fancier example:

“Negate amplitud

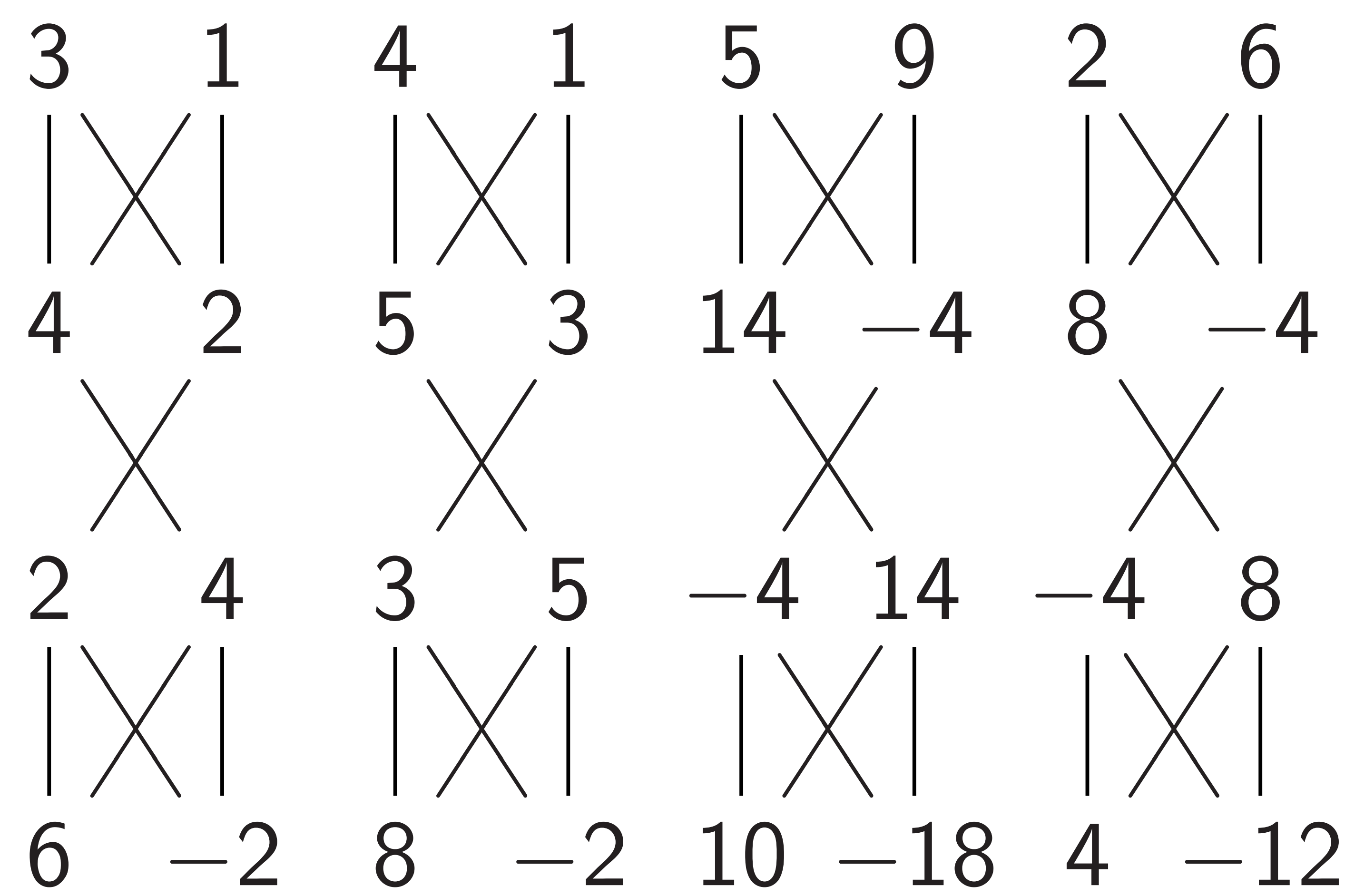
Assumes $q_2 = 0$:





Some uses of Hadamard gates

Hadamard_0, NOT_0, Hadamard_0:



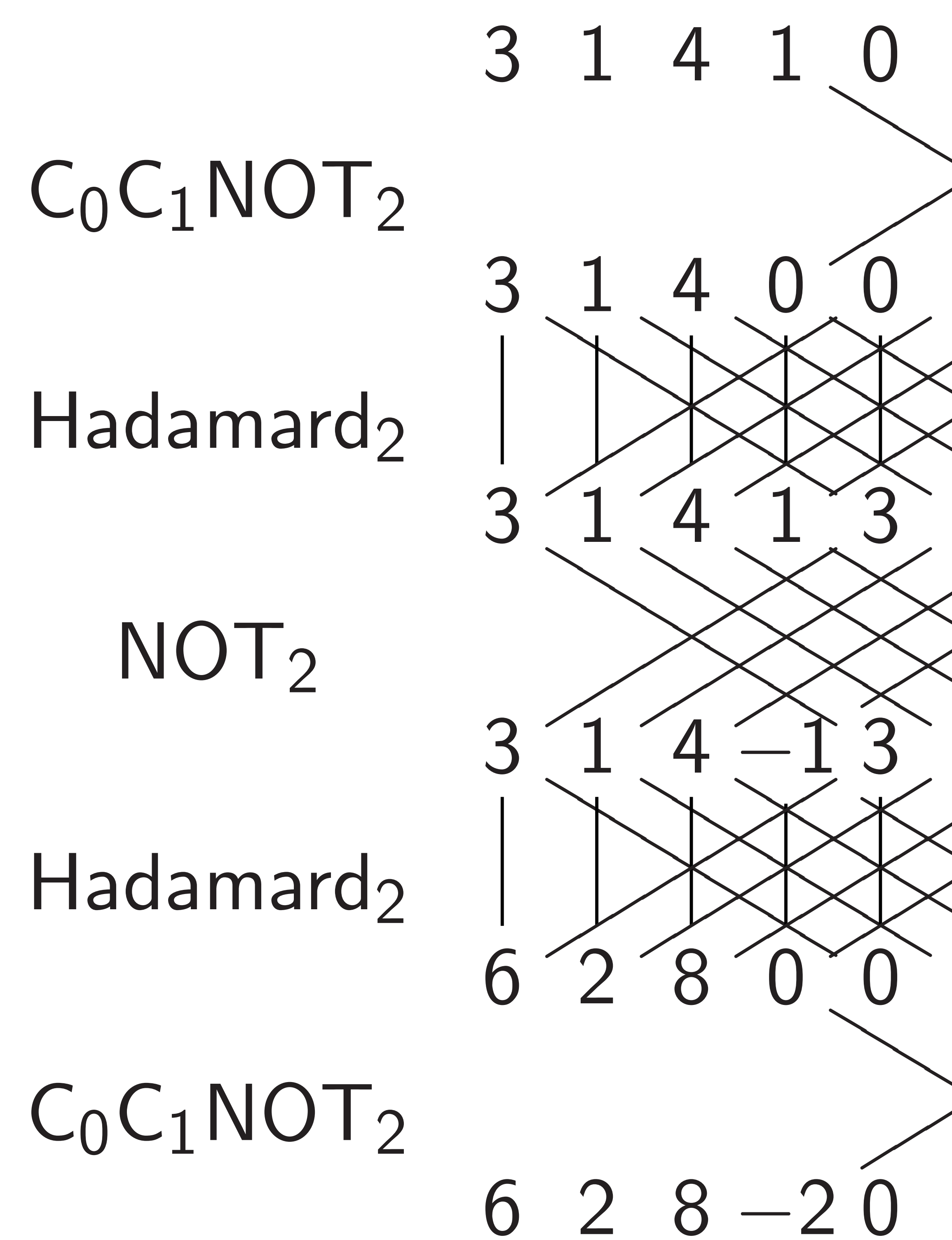
"Multiply each amplitude by 2."
This is not physically observable.

"Negate amplitude if q_0 is set."
No effect on measuring *now*.

Fancier example:

"Negate amplitude if $q_0 q_1$ is

Assumes $q_2 = 0$: "ancilla" c



Some uses of Hadamard gates

Hadamard$_0$, NOT$_0$, Hadamard$_0$:

$$\begin{array}{cccccccc}
3 & 1 & 4 & 1 & 5 & 9 & 2 & 6 \\
4 & 2 & 5 & 3 & 14 & -4 & 8 & -4 \\
2 & 4 & 3 & 5 & -4 & 14 & -4 & 8 \\
6 & -2 & 8 & -2 & 10 & -18 & 4 & -12
\end{array}$$

"Multiply each amplitude by 2."
This is not physically observable.

"Negate amplitude if q_0 is set."
No effect on measuring *now*.

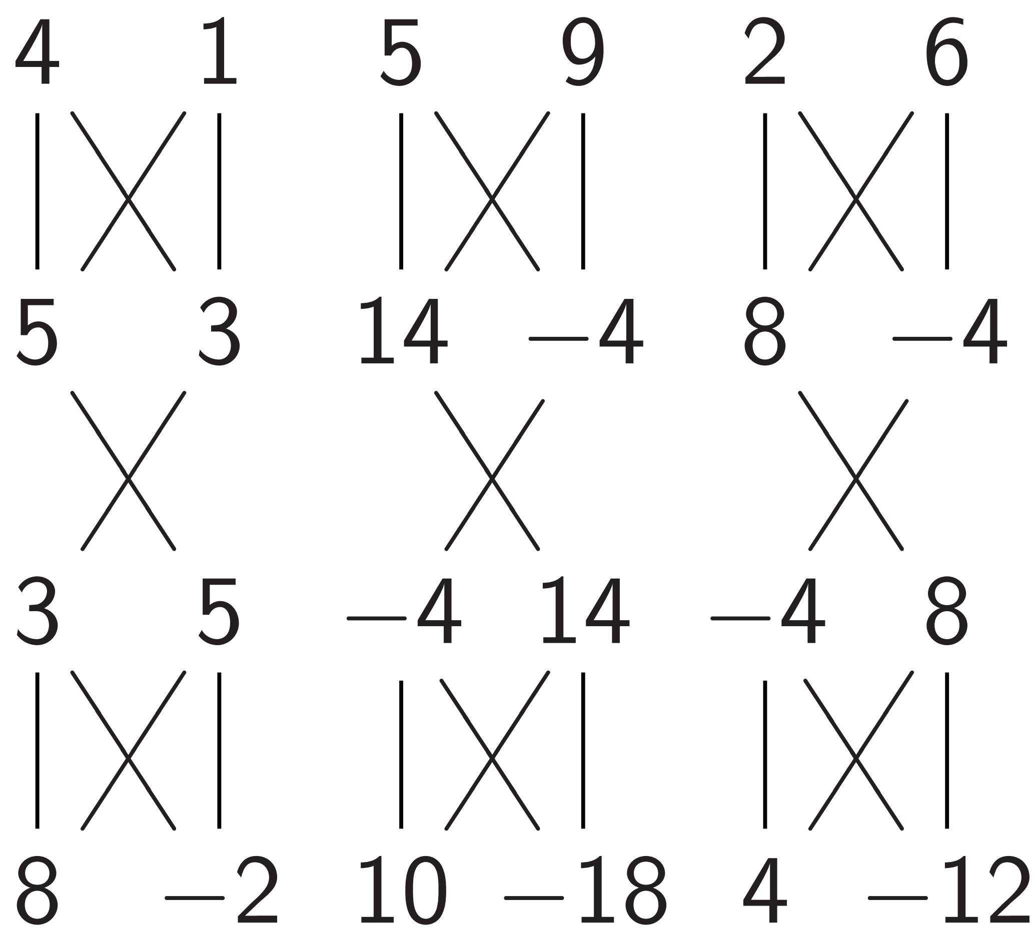
Fancier example:

"Negate amplitude if $q_0 q_1$ is set."
Assumes $q_2 = 0$: "ancilla" qubit.

$$\begin{array}{l|cccccccc}
 & 3 & 1 & 4 & 1 & 0 & 0 & 0 & 0 \\
\mathrm{C_0C_1NOT_2} & & & & & & & & \\
 & 3 & 1 & 4 & 0 & 0 & 0 & 0 & 1 \\
\mathrm{Hadamard_2} & & & & & & & & \\
 & 3 & 1 & 4 & 1 & 3 & 1 & 4 & -1 \\
\mathrm{NOT_2} & & & & & & & & \\
 & 3 & 1 & 4 & -1 & 3 & 1 & 4 & 1 \\
\mathrm{Hadamard_2} & & & & & & & & \\
 & 6 & 2 & 8 & 0 & 0 & 0 & 0 & -2 \\
\mathrm{C_0C_1NOT_2} & & & & & & & & \\
 & 6 & 2 & 8 & -2 & 0 & 0 & 0 & 0
\end{array}$$

ses of Hadamard gates

rd$_0$, NOT$_0$, Hadamard$_0$:



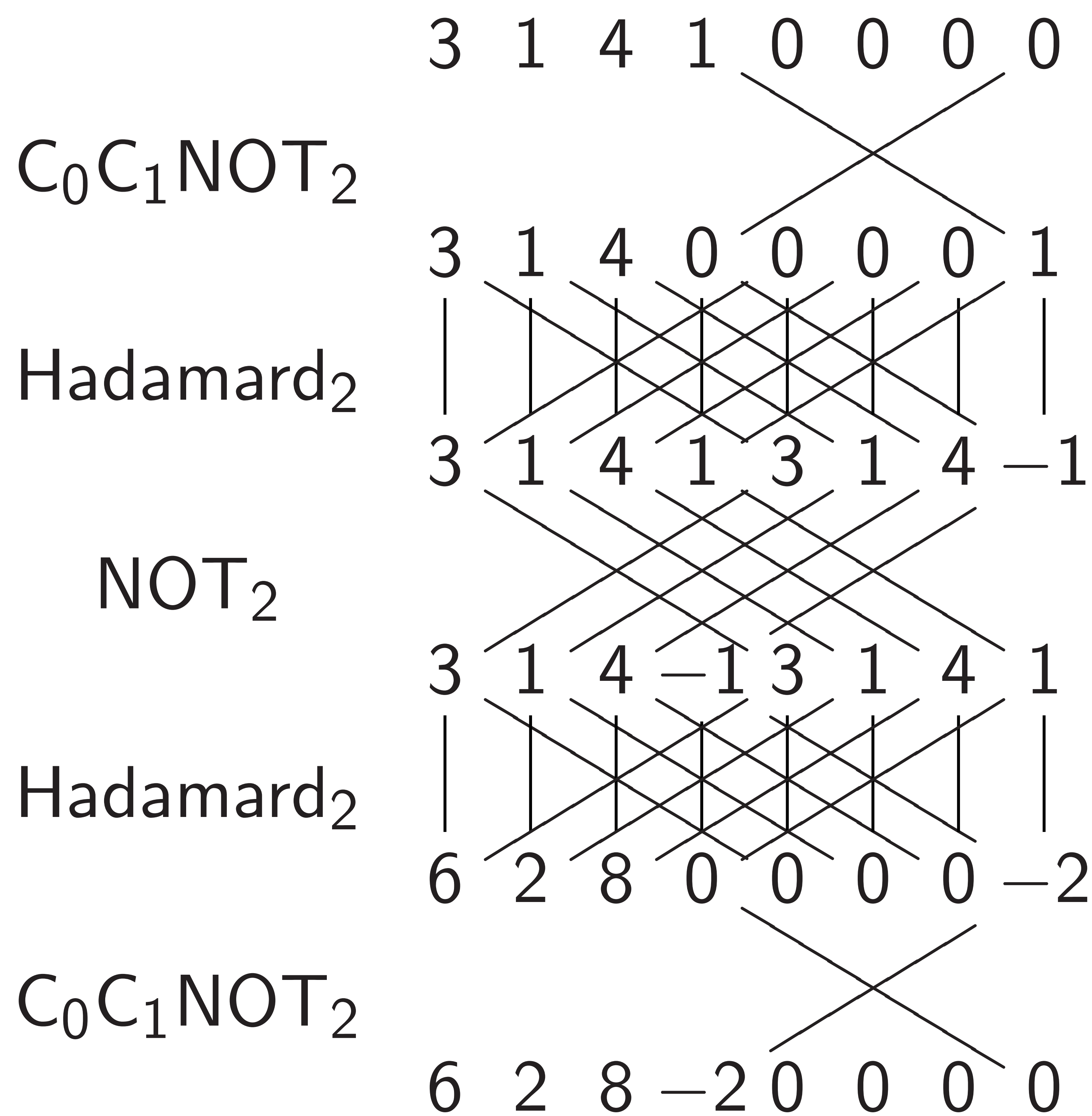
y each amplitude by 2.”
not physically observable.

amplitude if q_0 is set.”
t on measuring *now*.

Fancier example:

“Negate amplitude if $q_0 q_1$ is set.”

Assumes $q_2 = 0$: “ancilla” qubit.



Affects r
amplituc
$(3, 1, 4, 1$

amard gates

$_0$, Hadamard$_0$:

5 9 2 6

14 −4 8 −4

−4 14 −4 8

10 −18 4 −12

nplitude by 2.”

ally observable.

e if q_0 is set.”

uring *now*.

Fancier example:

“Negate amplitude if $q_0 q_1$ is set.”

Assumes $q_2 = 0$: “ancilla” qubit.

3 1 4 1 0 0 0 0

$C_0C_1NOT_2$

3 1 4 0 0 0 0 1

Hadamard$_2$

3 1 4 1 3 1 4 −1

NOT$_2$

3 1 4 −1 3 1 4 1

Hadamard$_2$

6 2 8 0 0 0 0 −2

$C_0C_1NOT_2$

6 2 8 −2 0 0 0 0

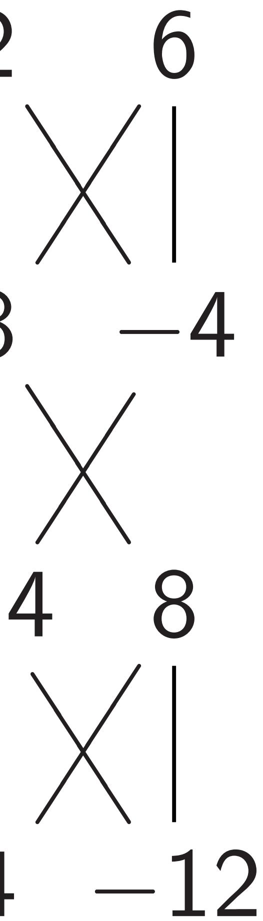
Affects measureme

amplitude around

$(3, 1, 4, 1) \mapsto (1.5,$

es

ard$_0$:



2.”

able.

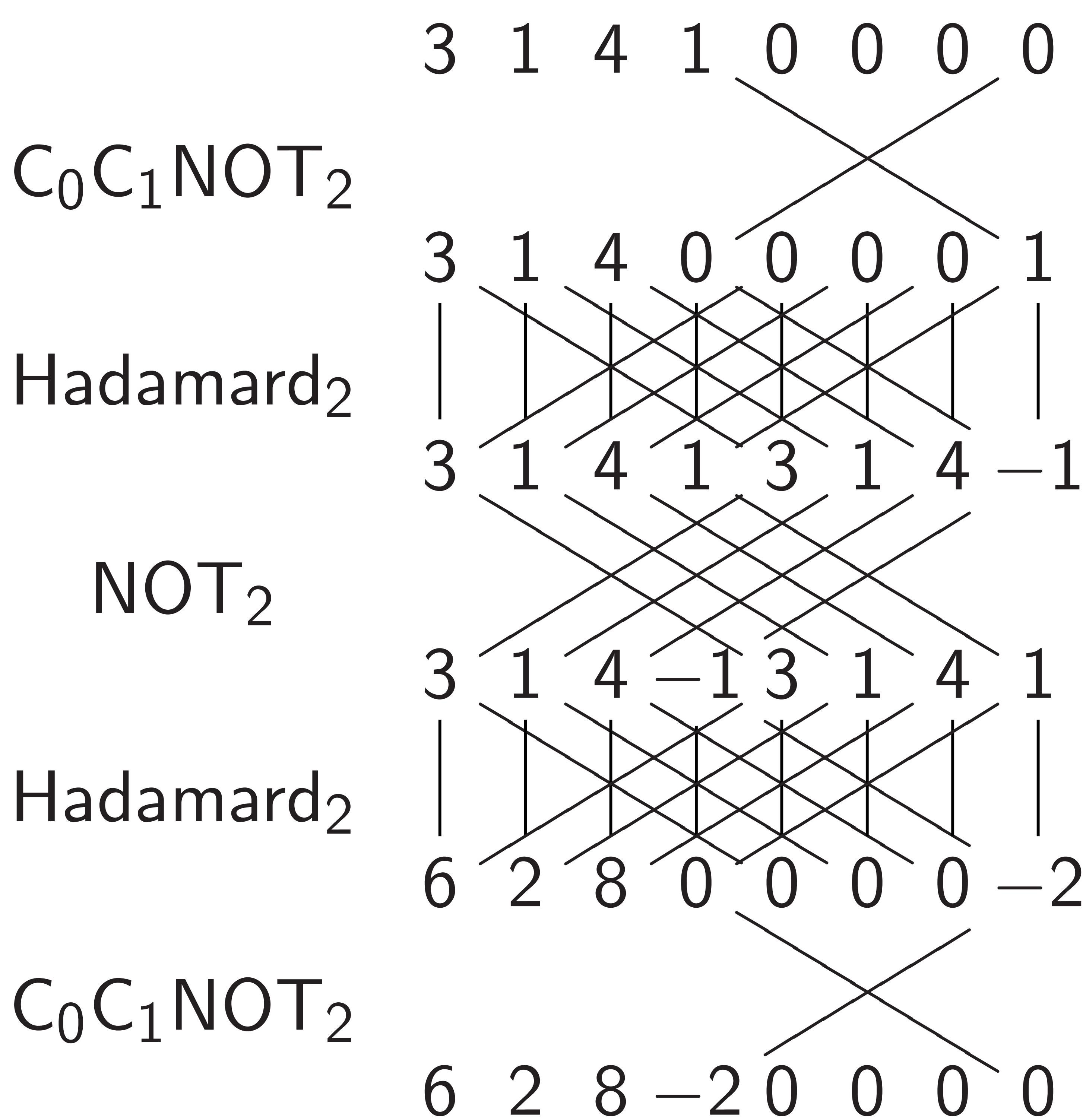
et.”

.

Fancier example:

“Negate amplitude if $q_0 q_1$ is set.”

Assumes $q_2 = 0$: “ancilla” qubit.



Affects measurements: “Neg

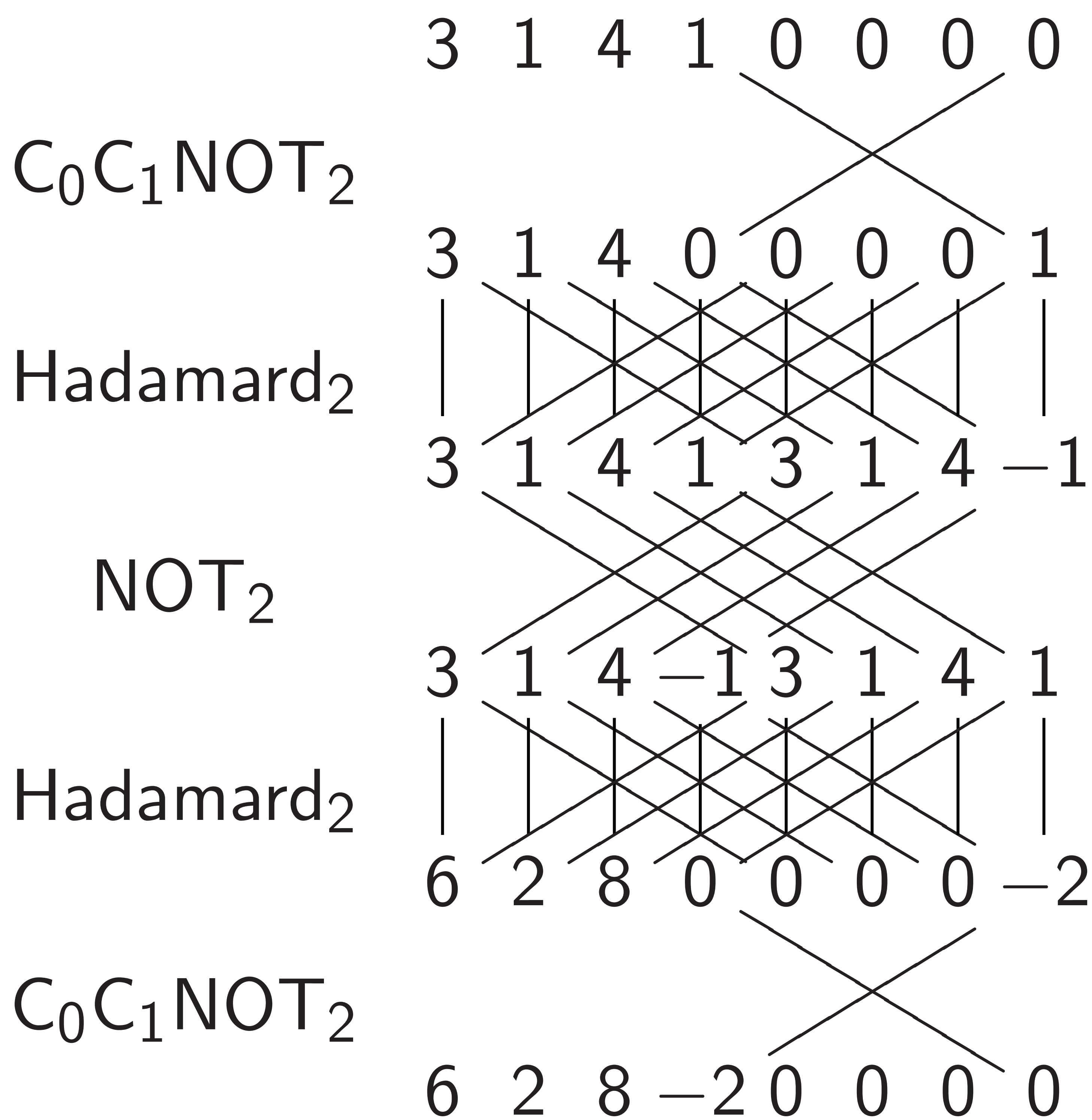
amplitude around its averag

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3$

Fancier example:

"Negate amplitude if $q_0 q_1$ is set."

Assumes $q_2 = 0$: "ancilla" qubit.



Affects measurements: "Negate amplitude around its average."

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

Fancier example:

"Negate amplitude if $q_0 q_1$ is set."

Assumes $q_2 = 0$: "ancilla" qubit.

	3	1	4	1	0	0	0	0
$\mathrm{C_0C_1NOT_2}$								
	3	1	4	0	0	0	0	1
$\mathrm{Hadamard_2}$								
	3	1	4	1	3	1	4	−1
$\mathrm{NOT_2}$								
	3	1	4	−1	3	1	4	1
$\mathrm{Hadamard_2}$								
	6	2	8	0	0	0	0	−2
$\mathrm{C_0C_1NOT_2}$								
	6	2	8	−2	0	0	0	0

Affects measurements: "Negate amplitude around its average."

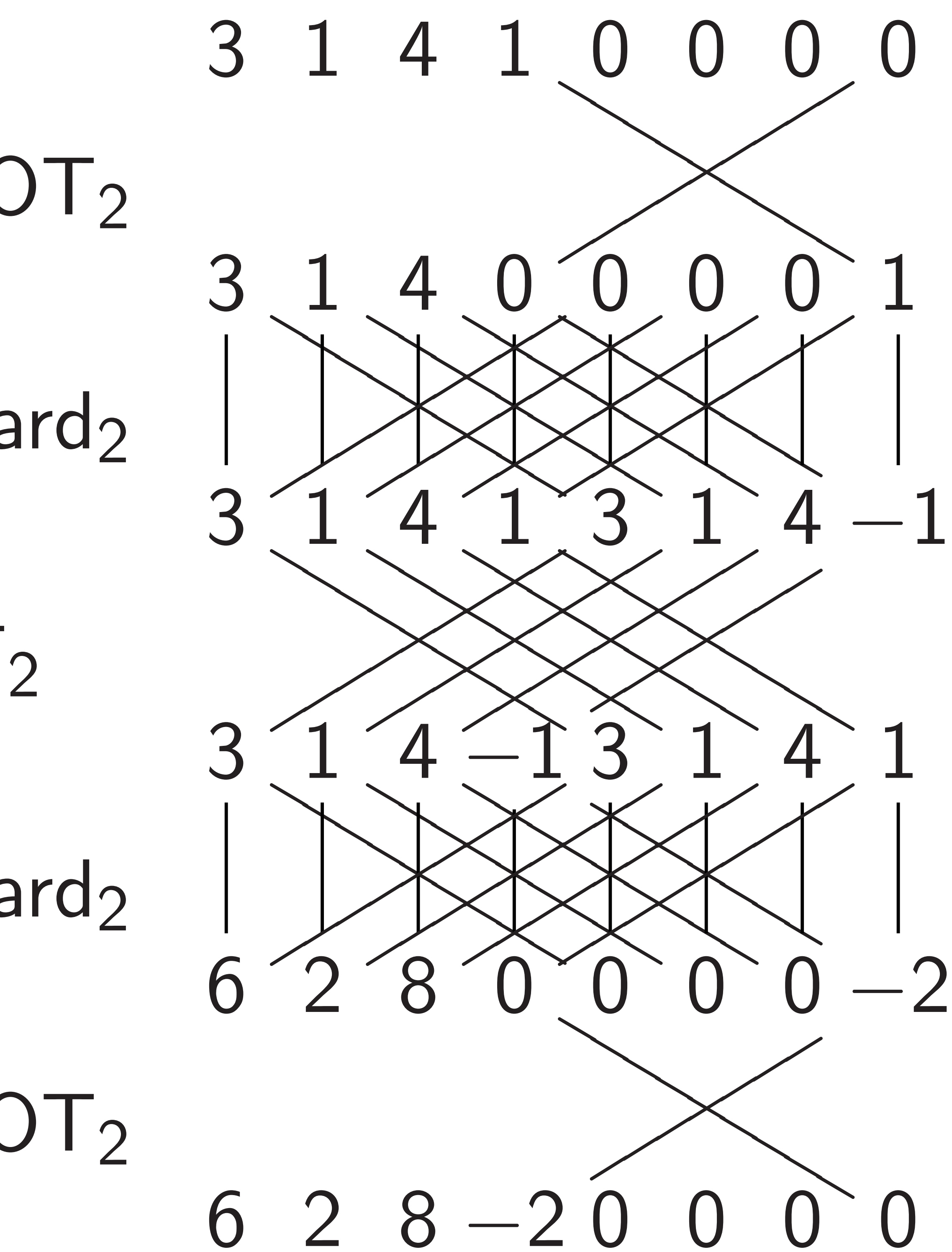
$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

	3	1	4	1	0	0	0	0
$\mathrm{H_0}$								
	4	2	5	3	0	0	0	0
$\mathrm{H_1}$								
	9	5	−1	−1	0	0	0	0
...								
	−9	5	−1	−1	0	0	0	0
$\mathrm{H_0}$								
	−4	−14	−2	0	0	0	0	0
$\mathrm{H_1}$								
	−6	−14	−2	−14	0	0	0	0

example:

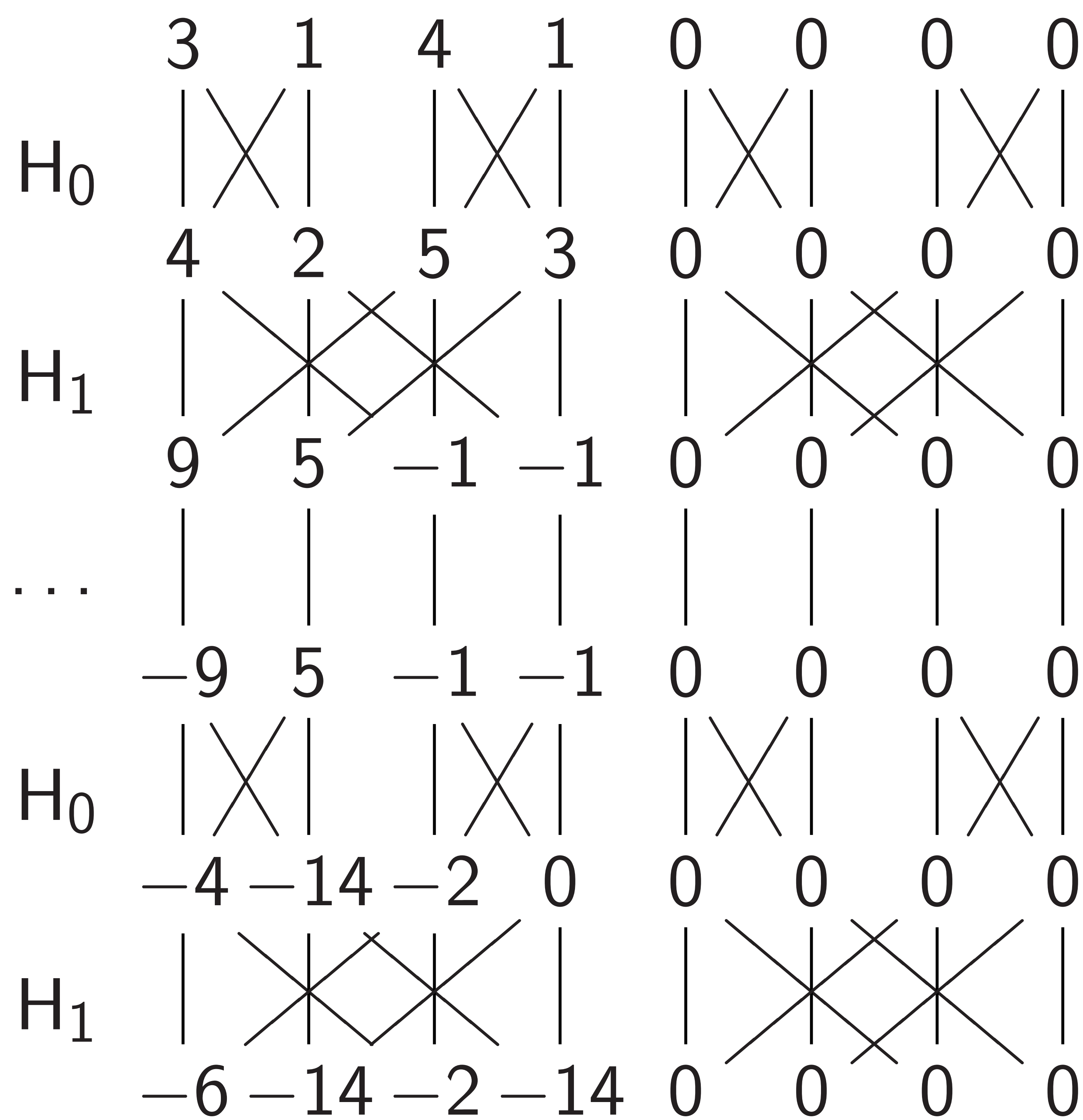
amplitude if $q_0 q_1$ is set."

s $q_2 = 0$: "ancilla" qubit.



Affects measurements: "Negate amplitude around its average."

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.



Simon's

Assumpt

- Given

can ef

- Nonze
- $f(u) =$
- f has

Goal: Fi

e if $q_0 q_1$ is set."
"ancilla" qubit.

4	1	0	0	0	0
4	0	0	0	0	1
4	1	3	1	4	−1
4	−1	3	1	4	1
8	0	0	0	0	−2
8	−2	0	0	0	0

Affects measurements: "Negate amplitude around its average."
$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

	3	1	4	1	0	0	0	0
H_0								
	4	2	5	3	0	0	0	0
H_1								
	9	5	−1	−1	0	0	0	0
...								
	−9	5	−1	−1	0	0	0	0
H_0								
	−4	−14	−2	0	0	0	0	0
H_1								
	−6	−14	−2	−14	0	0	0	0

Simon's algorithm

Assumptions:

- Given any $u \in \{$
  can efficiently co
- Nonzero $s \in \{0,$
- $f(u) = f(u \oplus s)$
- f has no other c

Goal: Figure out s

s set.”

qubit.

0	0	0
0	0	1
1	4	−1
1	4	1
0	0	−2
0	0	0

Affects measurements: “Negate amplitude around its average.”

$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

	3	1	4	1	0	0	0	0
H_0								
	4	2	5	3	0	0	0	0
H_1								
	9	5	−1	−1	0	0	0	0
. . .								
	−9	5	−1	−1	0	0	0	0
H_0								
	−4	−14	−2	0	0	0	0	0
H_1								
	−6	−14	−2	−14	0	0	0	0

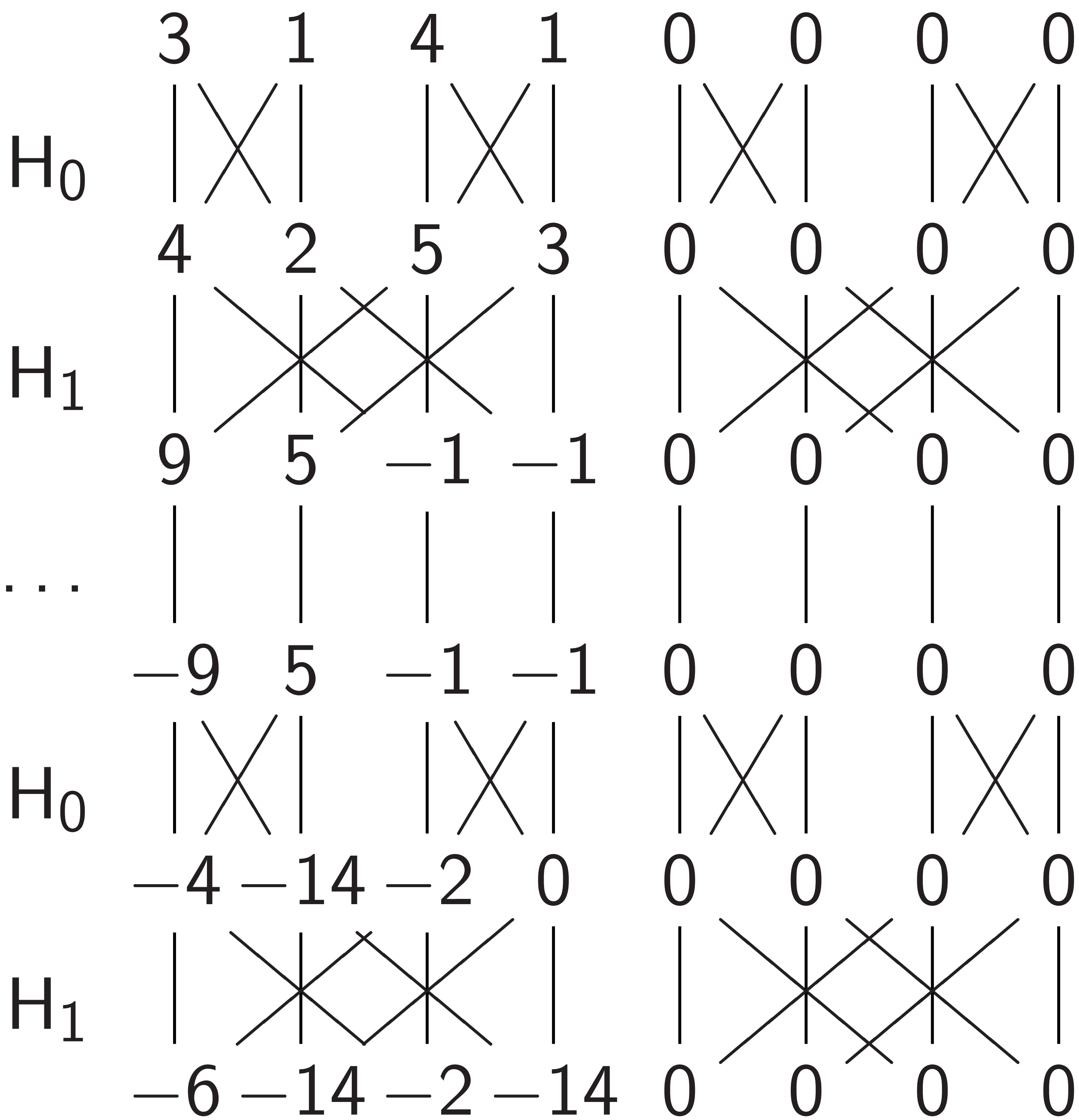
<u>Simon’s algorithm</u>

Assumptions:

- Given any $u \in \{0, 1\}^n$, can efficiently compute $f($
- Nonzero $s \in \{0, 1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Affects measurements: "Negate amplitude around its average."
$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.



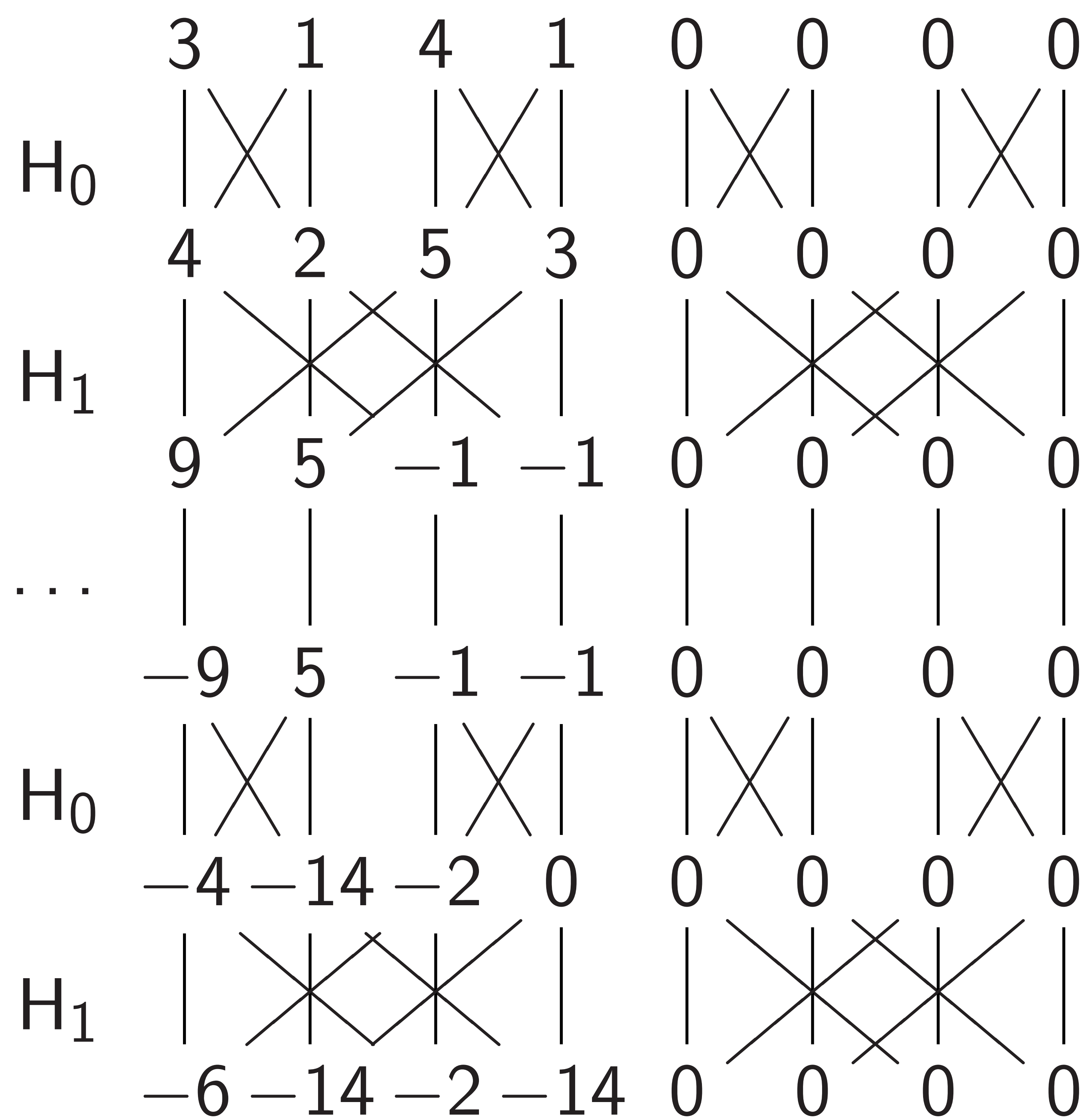
Simon's algorithm

Assumptions:

- Given any $u \in \{0, 1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0, 1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Affects measurements: "Negate amplitude around its average."
$(3,1,4,1) \mapsto (1.5,3.5,0.5,3.5)$.



Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Affects measurements: "Negate amplitude around its average."
$(3, 1, 4, 1) \mapsto (1.5, 3.5, 0.5, 3.5)$.

	3	1	4	1	0	0	0	0
H_0								
	4	2	5	3	0	0	0	0
H_1								
	9	5	−1	−1	0	0	0	0
$\cdots$								
	−9	5	−1	−1	0	0	0	0
H_0								
	−4	−14	−2	0	0	0	0	0
H_1								
	−6	−14	−2	−14	0	0	0	0

Simon's algorithm

Assumptions:

- Given any $u \in \{0, 1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0, 1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

measurements: "Negate

de around its average."

1) $\mapsto$ (1.5, 3.5, 0.5, 3.5).

1	4	1	0	0	0	0
2	5	3	0	0	0	0
5	−1	−1	0	0	0	0
5	−1	−1	0	0	0	0
−14	−2	0	0	0	0	0
−14	−2	−14	0	0	0	0

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example

Step 1.

1, 0, 0,

0, 0, 0,

0, 0, 0,

0, 0, 0,

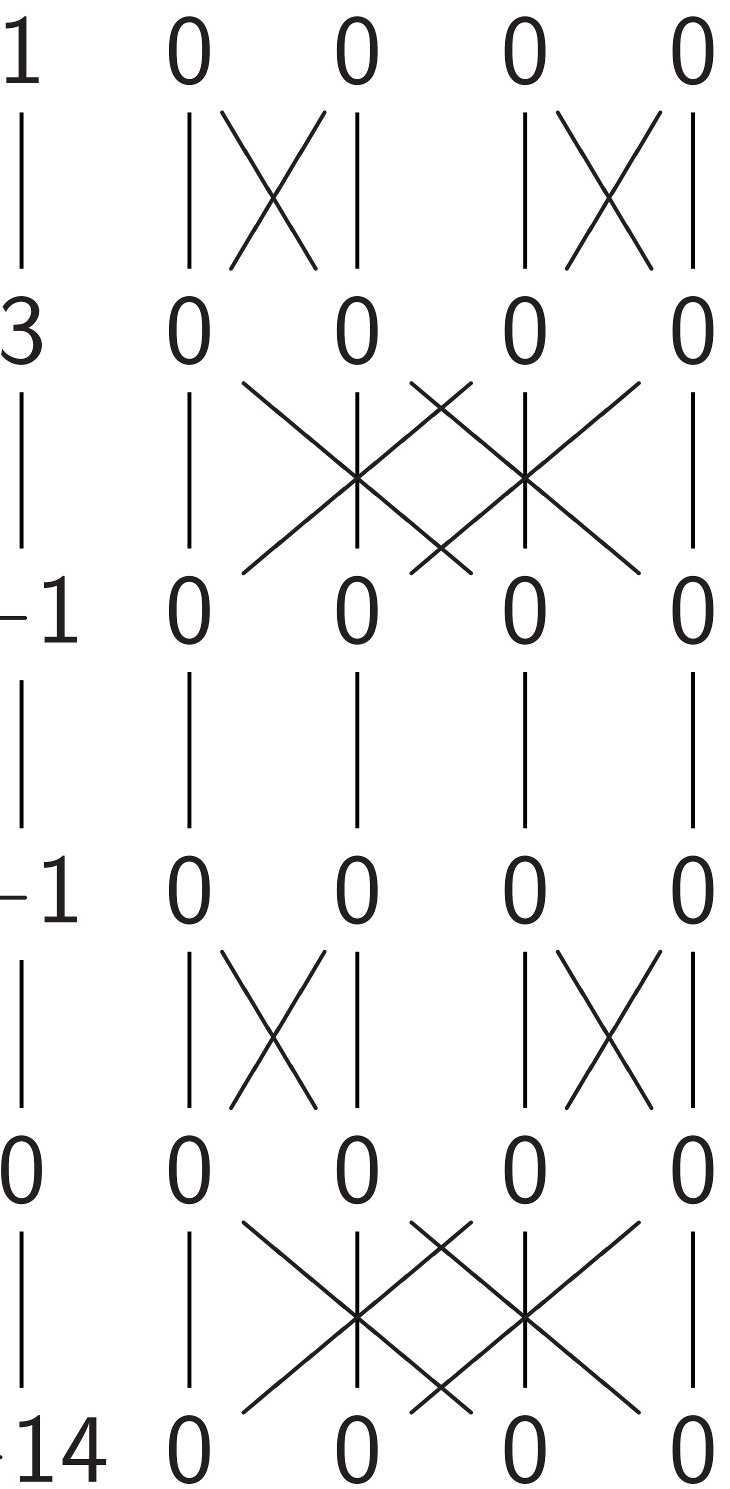
0, 0, 0,

0, 0, 0,

0, 0, 0,

0, 0, 0,

ents: "Negate
its average."
3.5, 0.5, 3.5).



Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon

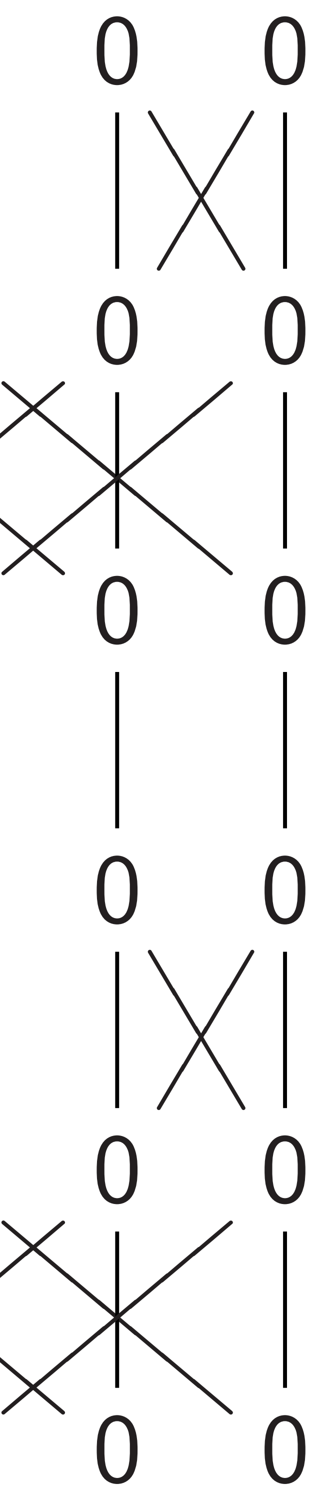
Step 1. Set up pu

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

gate

e.”

.5).



Simon’s algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon’s algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon’s algorith

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.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 1. Set up pure zero state:

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 2. Hadamard$_0$:

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 3. Hadamard$_1$:

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 4. Hadamard$_2$:

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.

Each column is a parallel universe.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5. C_0NOT_3:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5b. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5c. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5d. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5e. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5f. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5g. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5h. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5i. More shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5j. Final shuffling:

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

Each column is a parallel universe performing its own computations.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 5j. Final shuffling:

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

Each column is a parallel universe performing its own computations.
Surprise: u and $u \oplus 101$ match.

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 6. Hadamard$_0$:

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

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 7. Hadamard$_1$:

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

$1, \overline{1}, \overline{1}, 1, 1, 1, \overline{1}, \overline{1},$

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

$1, 1, \overline{1}, \overline{1}, 1, \overline{1}, \overline{1}, 1,$

$1, \overline{1}, 1, \overline{1}, 1, 1, 1, 1,$

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

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

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

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

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

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

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

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

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

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

Simon's algorithm

Assumptions:

- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Traditional algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum computations of f.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

algorithm

tions:

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

ficiently compute $f(u)$.

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

$= f(u \oplus s)$ for all u.

no other collisions.

gure out s.

nal algorithm to find s:

e f for many inputs,

find collision.

algorithm finds s with

ntum computations of f.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

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

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

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

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

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

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

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat t

$0, 1\}^n$,

ompute $f(u)$.

$1\}^n$.

for all u.

collisions.

s.

hm to find s:

ny inputs,

ion.

finds s with

outations of f.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

$0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,$

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

$0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,$

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

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

$0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,$

$0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,\ 0,$

$2,\ 0,\ 2,\ 0,\ 0,\ 2,\ 0,\ 2.$

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure o

u).

l s:

ith

of f.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

0, 0, 0, 0, 0, 0, 0, 0,

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

0, 0, 0, 0, 0, 0, 0, 0,

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

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

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, 2, 0, 0, 2, 0, 2.

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

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

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

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

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

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

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

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

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

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

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

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

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

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

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

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

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

Example of Simon's algorithm

Step 8. Hadamard$_2$:

0, 0, 0, 0, 0, 0, 0, 0,

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

0, 0, 0, 0, 0, 0, 0, 0,

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

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

0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0,

2, 0, 2, 0, 0, 2, 0, 2.

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

e of Simon's algorithm

Hadamard$_2$:

0, 0, 0, 0, 0,

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

0, 0, 0, 0, 0,

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

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

0, 0, 0, 0, 0,

0, 0, 0, 0, 0,

0, 0, 2, 0, 2.

Measure. Obtain some
tion about the surprise: a
vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

Grover's

Assume:
has $f(s)$

Traditio
compute
hope to
Success
until #i

's algorithm

d$_2$:

0, 0,

0, 2,

0, 0,

0, $\overline{2}$,

0, $\overline{2}$,

0, 0,

0, 0,

0, 2.

Obtain some
the surprise: a
hogonal to 101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

Grover's algorithm

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

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

m

me

se: a

101.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

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

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

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.

Repeat to figure out 101.

Generalize Step 5 to any function $u \mapsto f(u)$ with $f(u) = f(u \oplus s)$. "Usually" algorithm figures out s.

Shor's algorithm replaces $\oplus$ with more general $+$ operation. Many spectacular applications.

e.g. Shor finds "random" s with $2^u \bmod N = 2^{u+s} \bmod N$. Easy to factor N using this.

e.g. Shor finds "random" s, t with $4^u 9^v \bmod p = 4^{u+s} 9^{v+t} \bmod p$. Easy to compute discrete logs.

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.

to figure out 101.

ze Step 5 to any function
u) with $f(u) = f(u \oplus s)$.
" algorithm figures out s.

lgorithm replaces $\oplus$
re general $+$ operation.
pectacular applications.

r finds "random" s with
$N = 2^{u+s} \bmod N$.
factor N using this.

r finds "random" s, t with
od $p = 4^{u+s} 9^{v+t} \bmod p$.
compute discrete logs.

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

ut 101.

to any function
$u) = f(u \oplus s)$.
m figures out s.

eplaces $\oplus$
$+$ operation.
applications.

ndom" s with
mod N.
using this.

ndom" s, t with
$^{-s}g^{v+t}$ mod p.
discrete logs.

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

nction

$\oplus s)$.

out s.

ion.

ns.

with

t with

od p.

gs.

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 u.

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 u.

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 u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

Grover's algorithm

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

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

Grover's algorithm takes only $2^{n/2}$
reversible computations of f.
Typically: reversibility overhead
is small enough that this
easily beats traditional algorithm.

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

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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

Grover's algorithm

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

Traditional algorithm to find s: compute f for many inputs, hope to find output 0. Success probability is very low until #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 u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ 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.

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 u.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ 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 u.

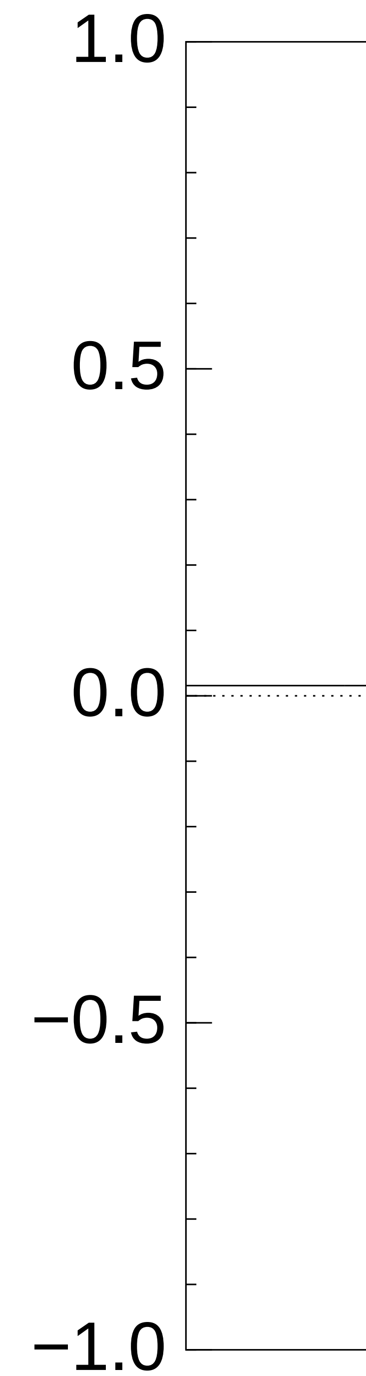
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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
onal algorithm.

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

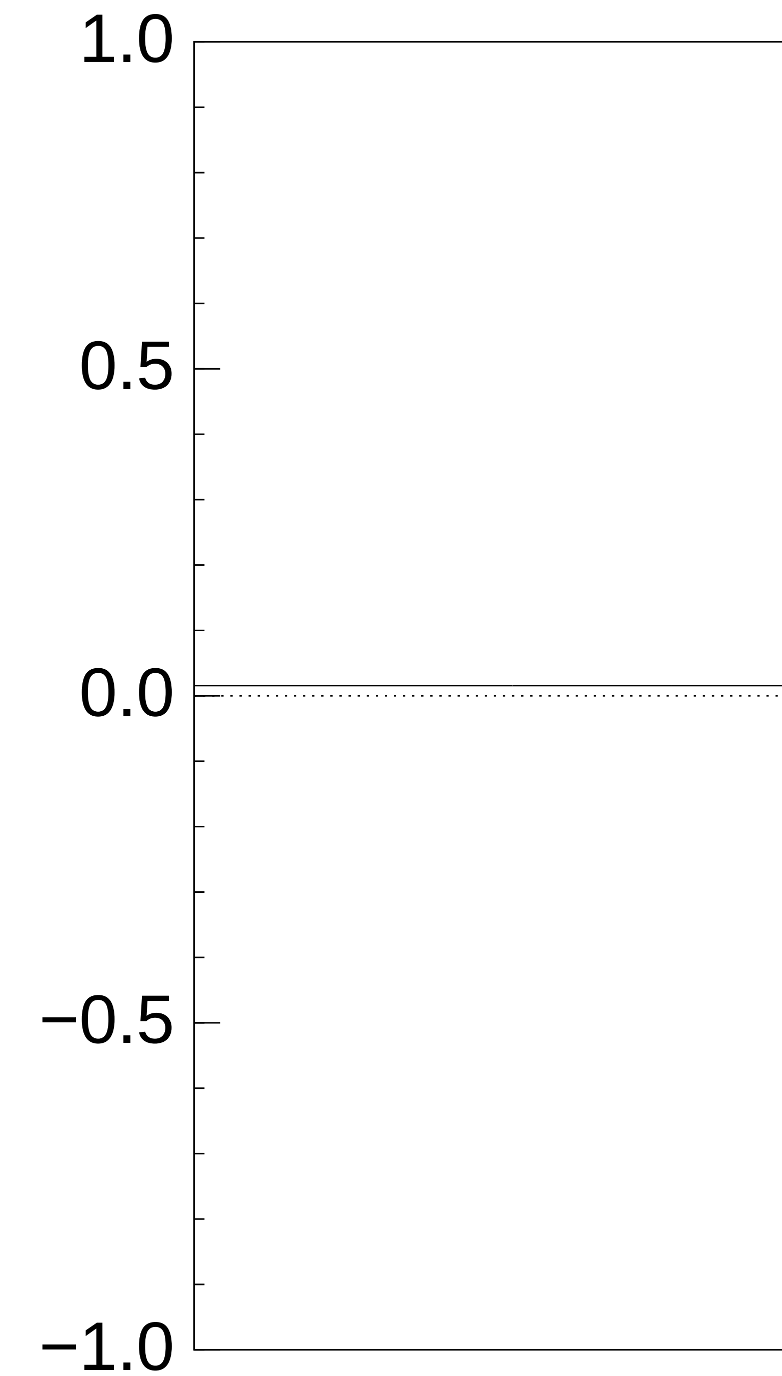
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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:



l s:

ow

?.

y $2^{n/2}$

?.

ead

ithm.

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

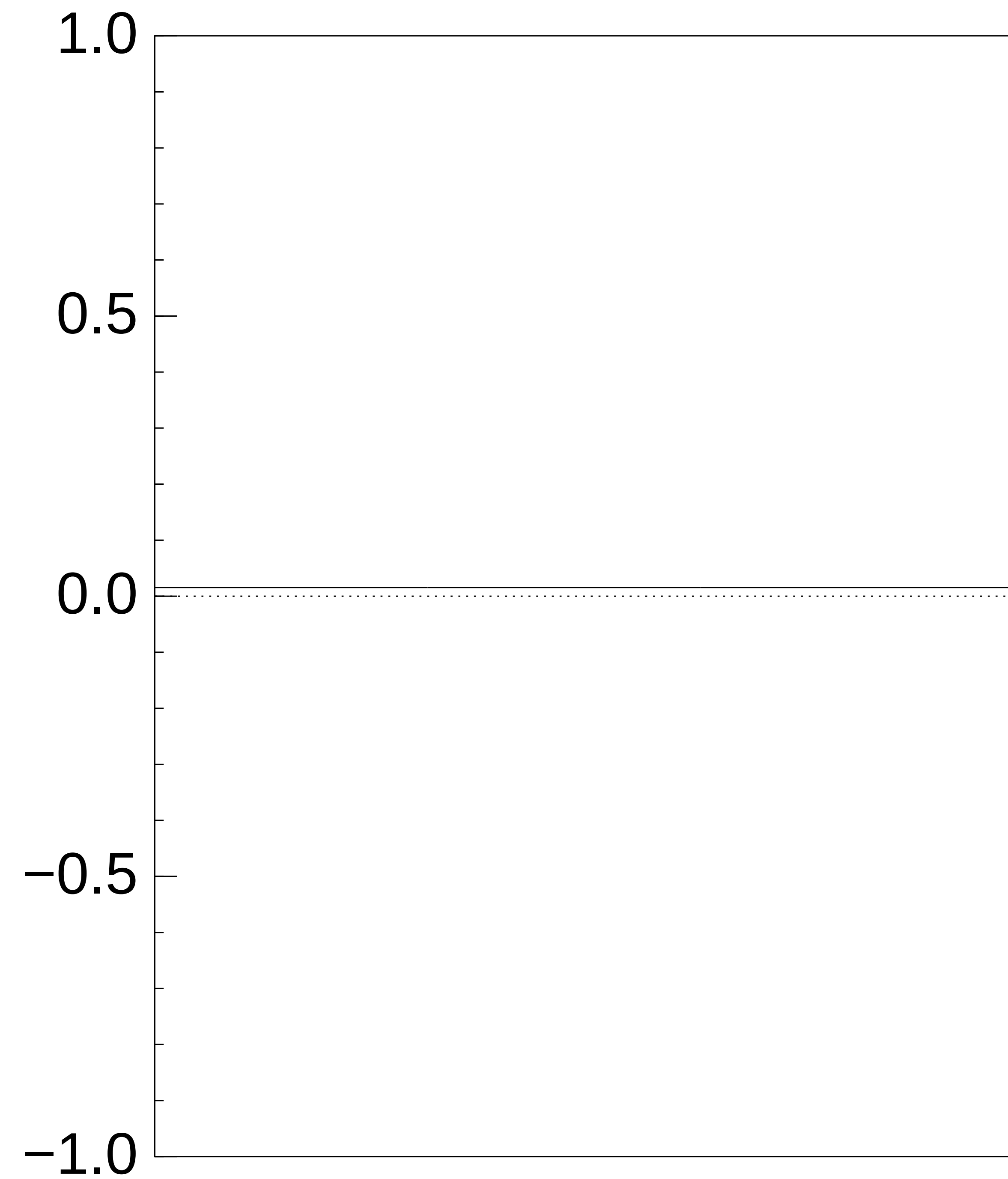
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$
for an example with $n = 12$
after 0 steps:



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

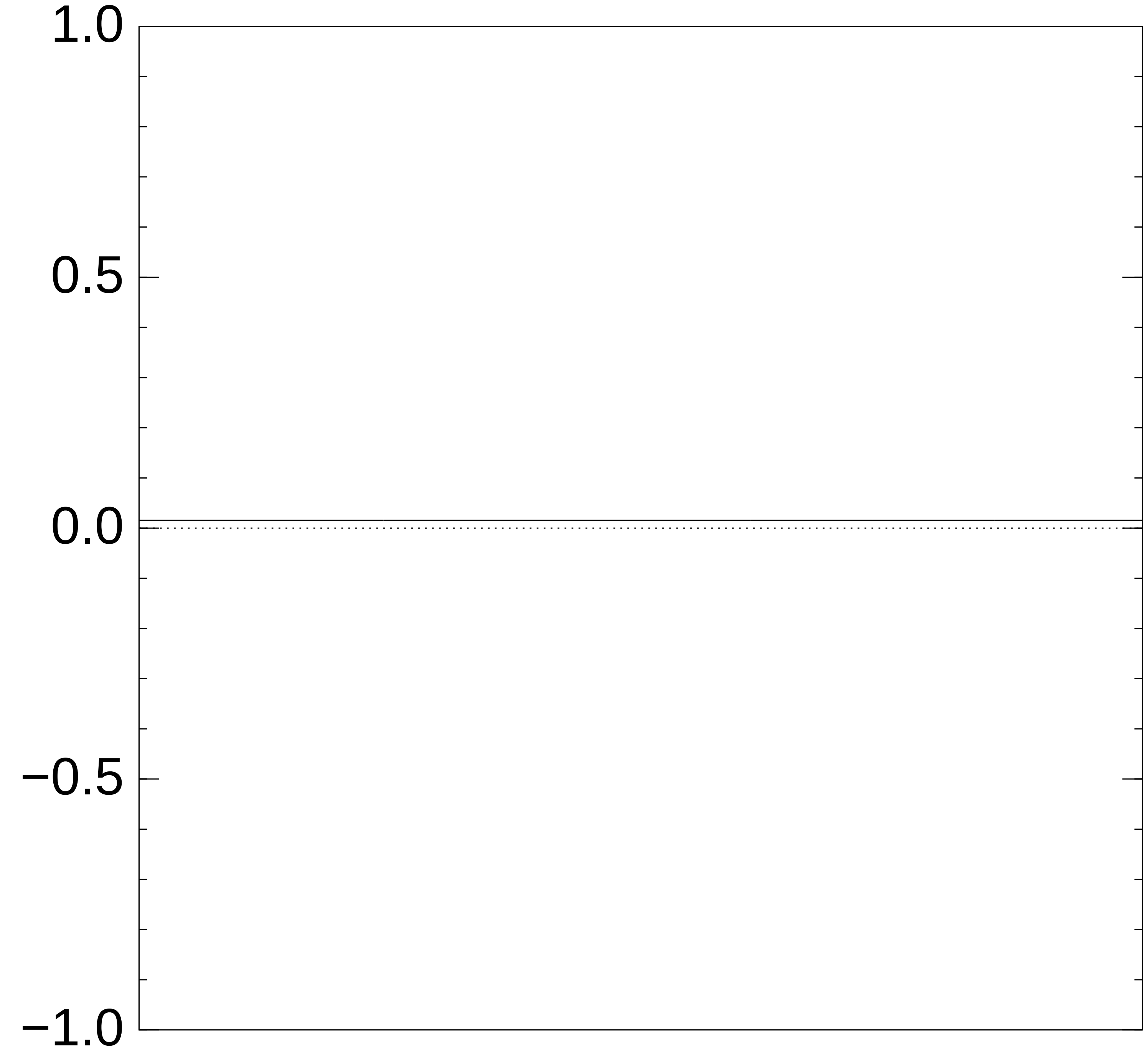
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after 0 steps:



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

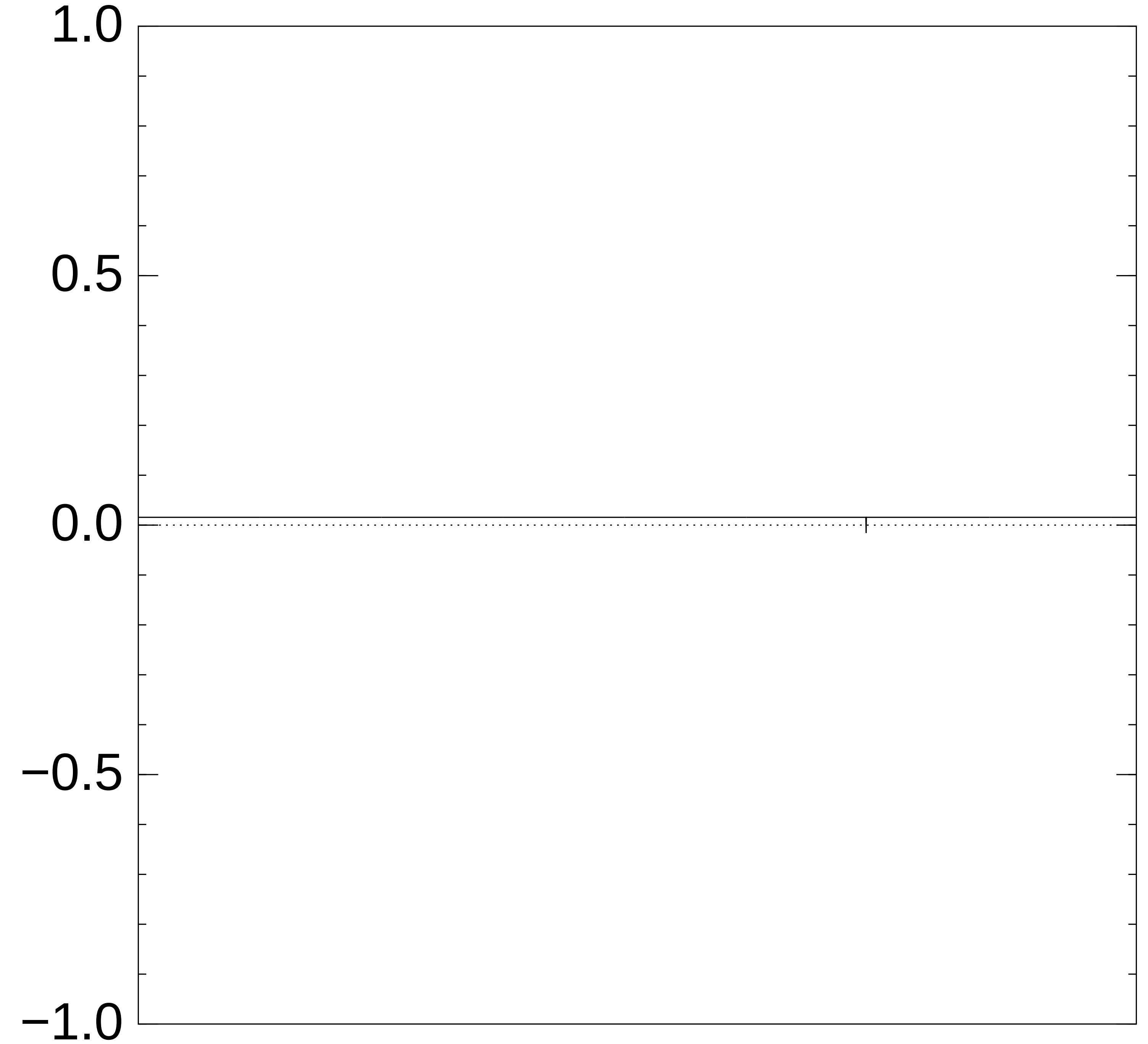
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after Step 1:



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

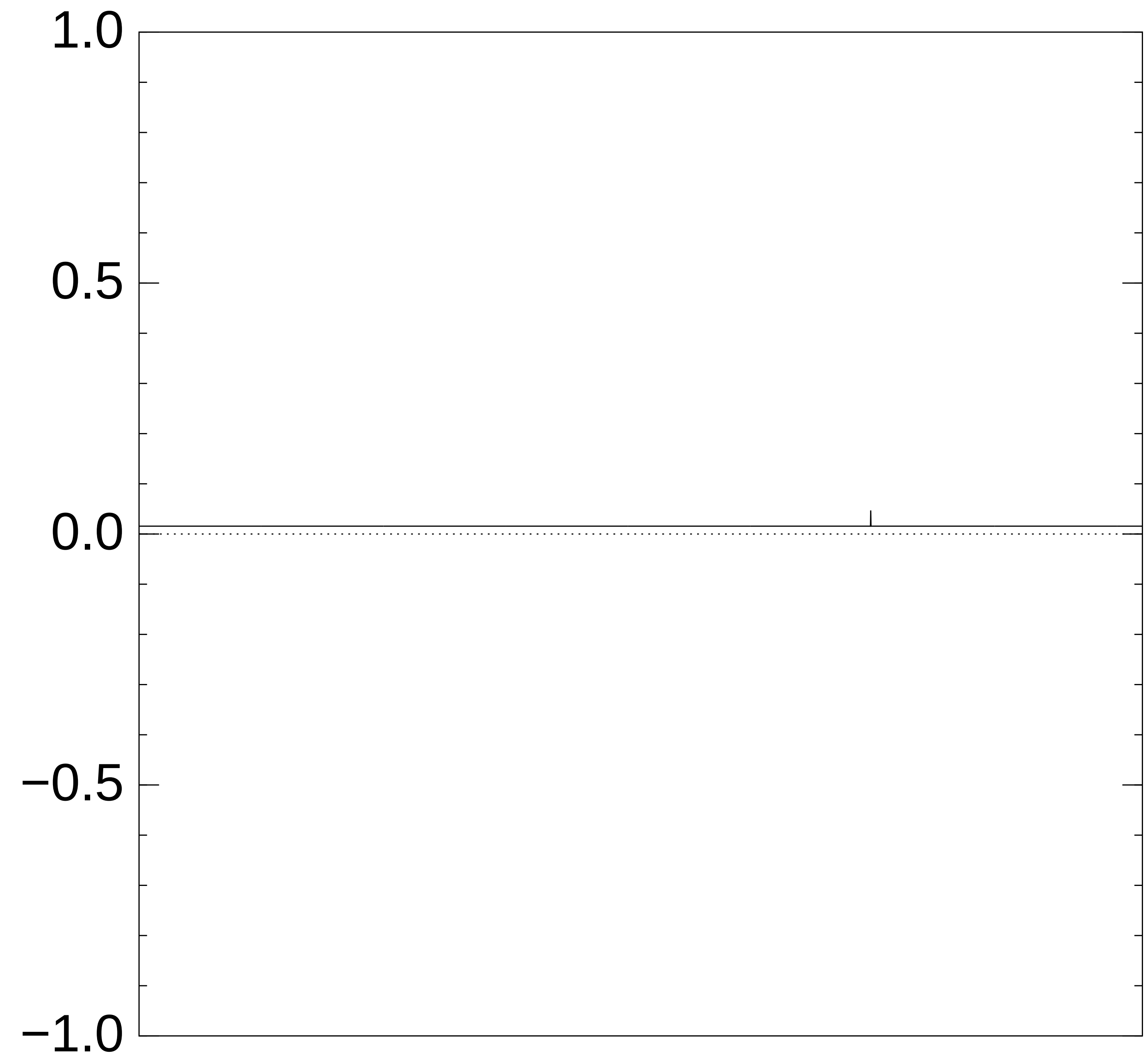
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after Step 1 + Step 2:



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

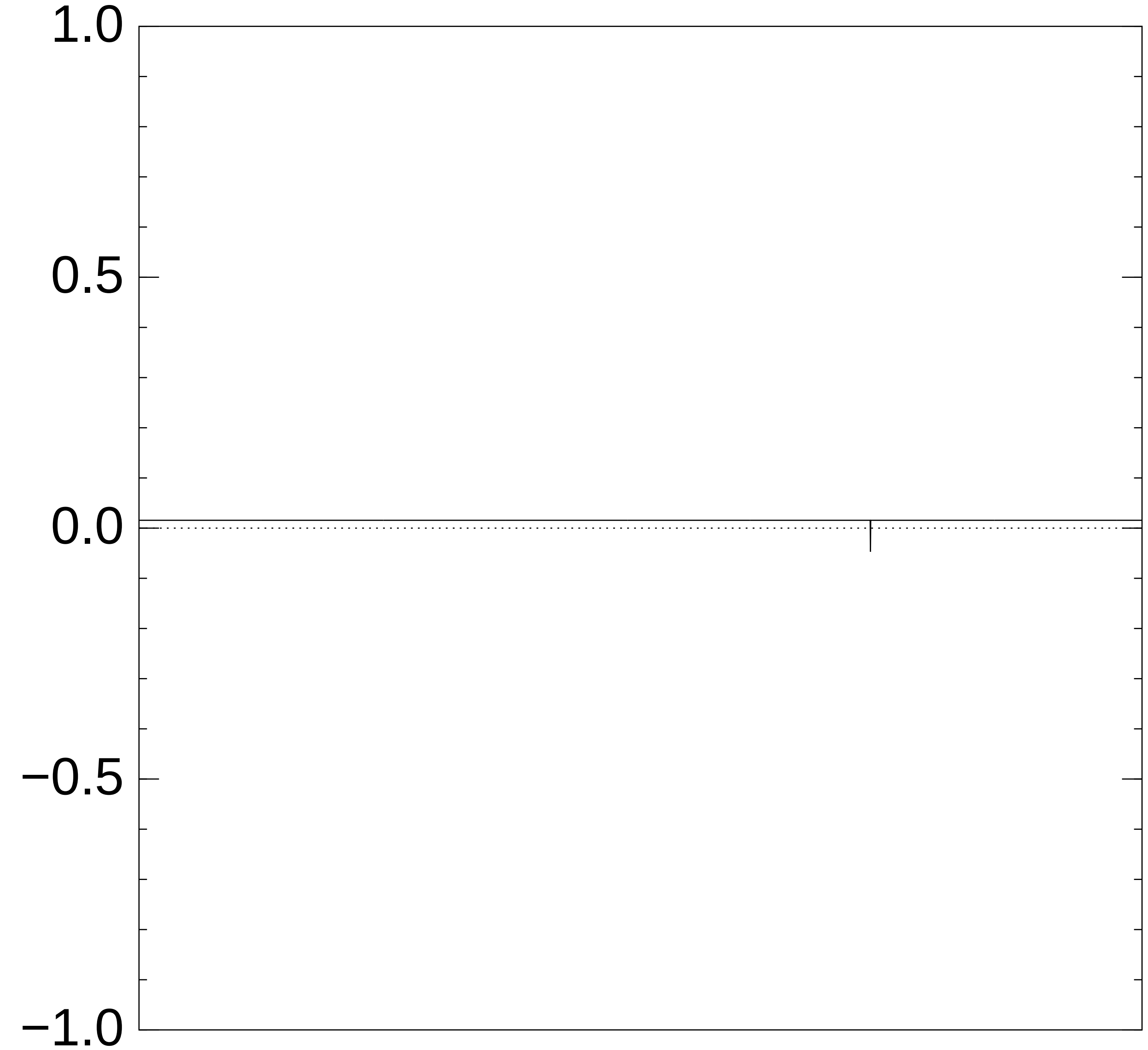
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after Step 1 + Step 2 + Step 1:



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

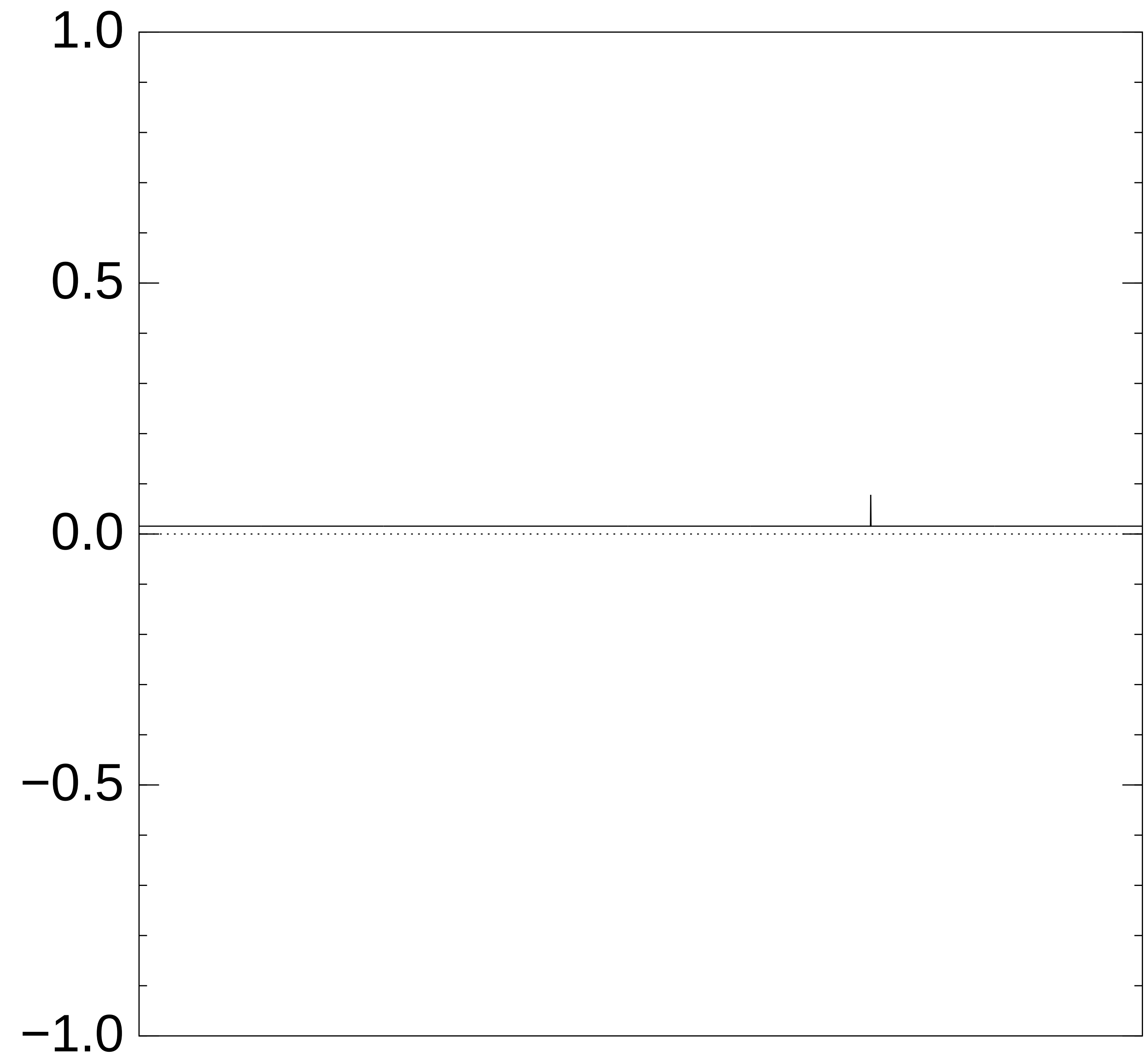
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $2 \times$ (Step 1 + Step 2):



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

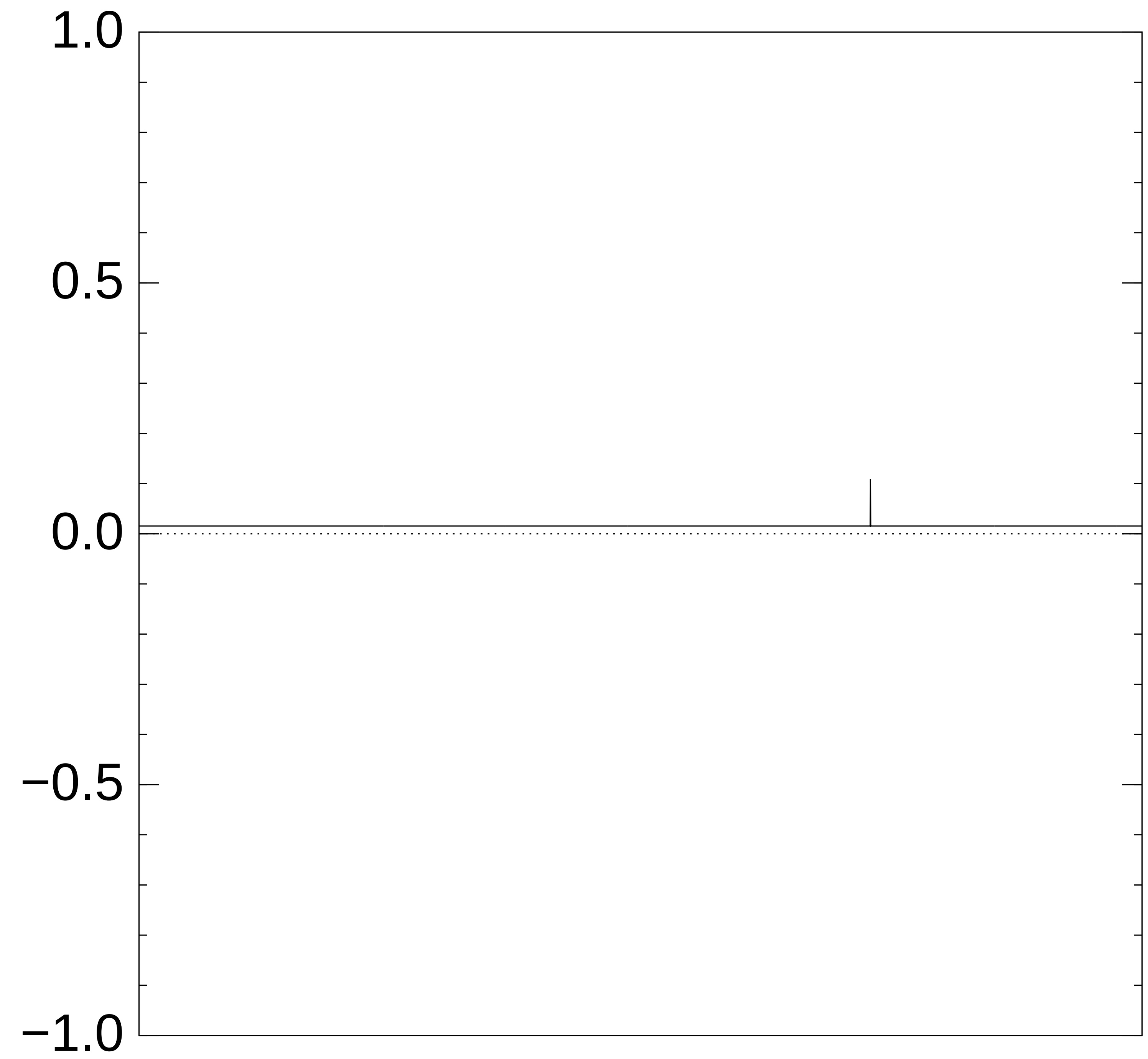
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $3 \times$ (Step 1 + Step 2):



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

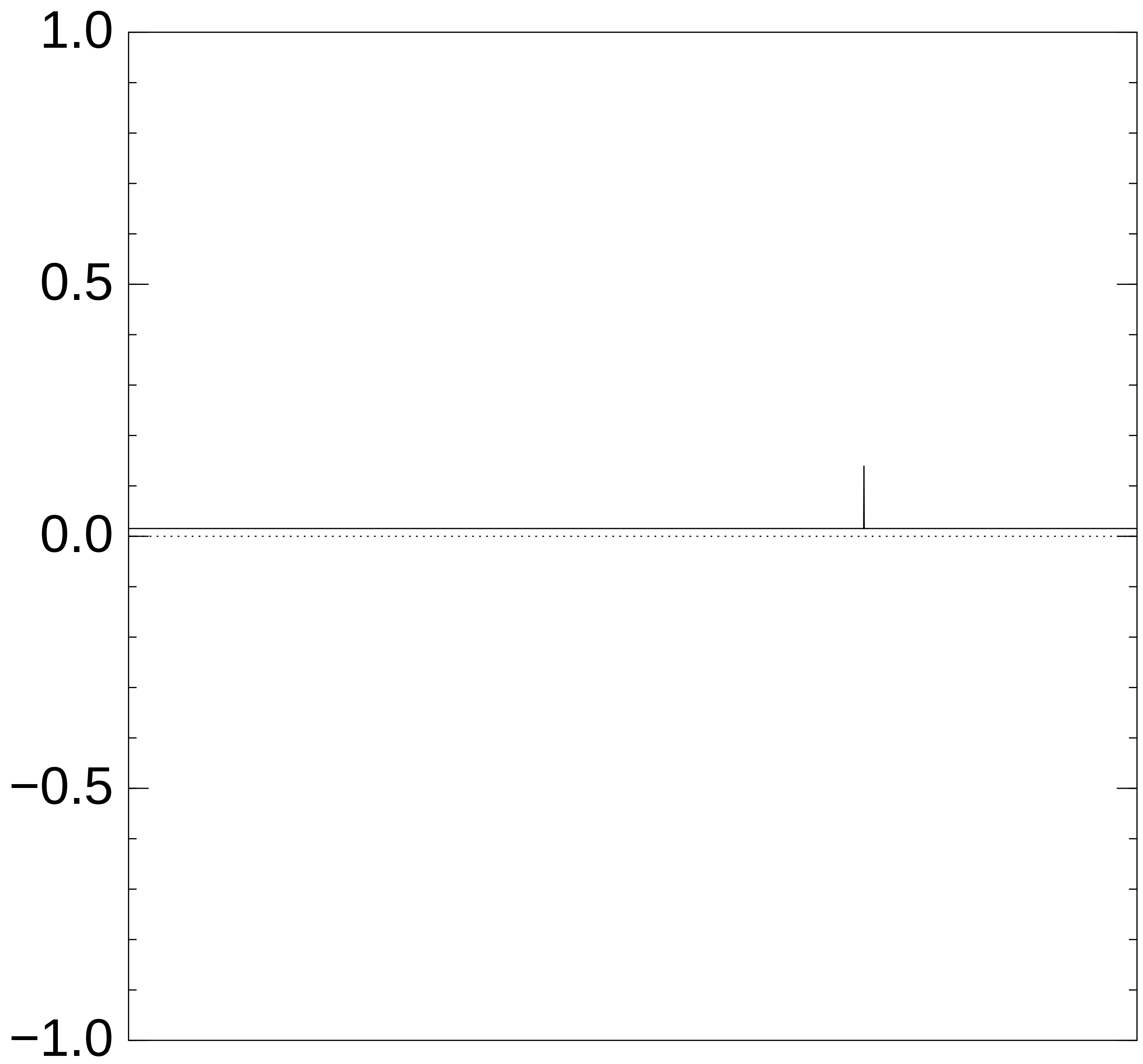
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $4 \times$ (Step 1 + Step 2):



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

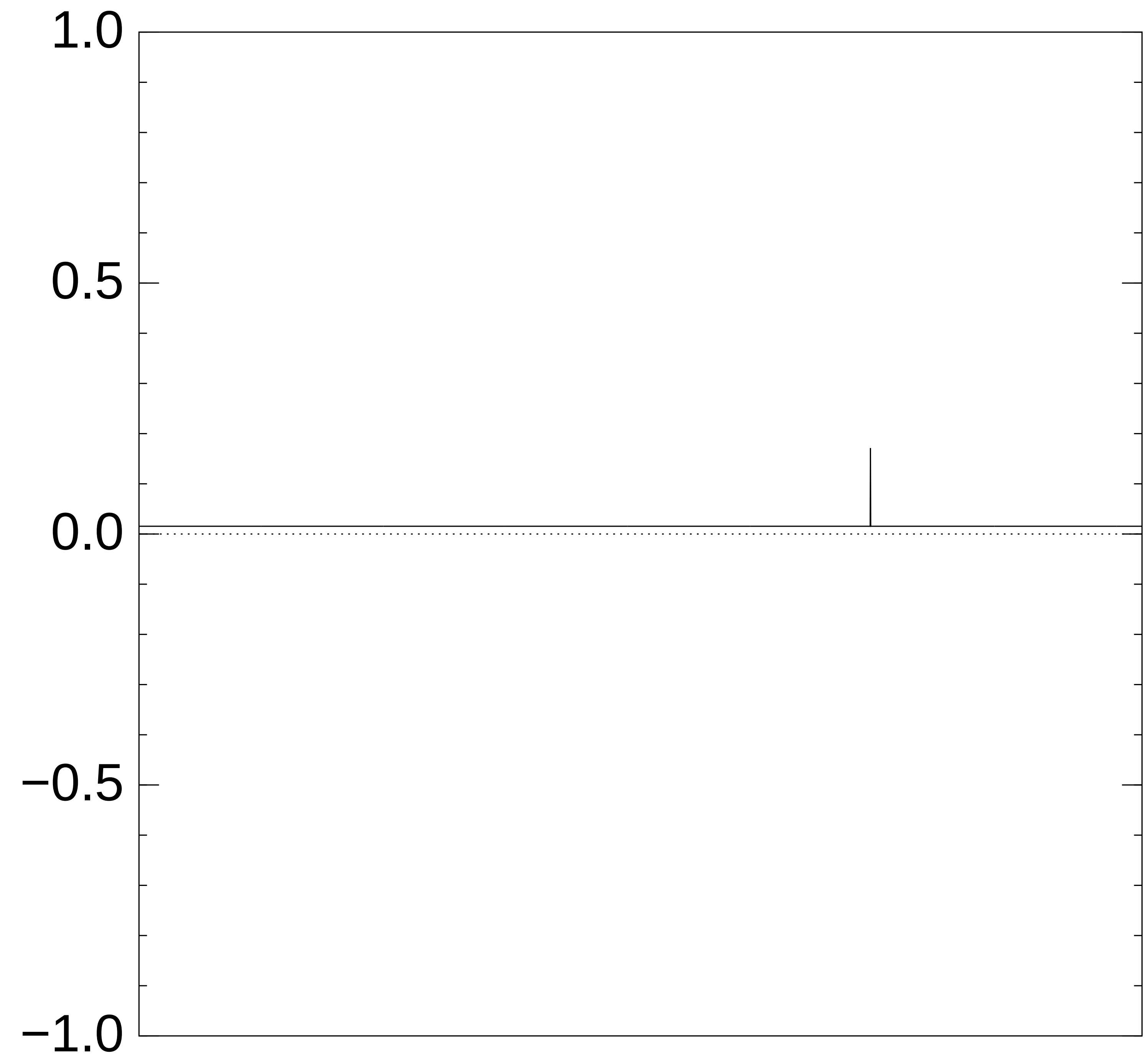
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$
for an example with $n = 12$
after $5 \times$ (Step 1 + Step 2):



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

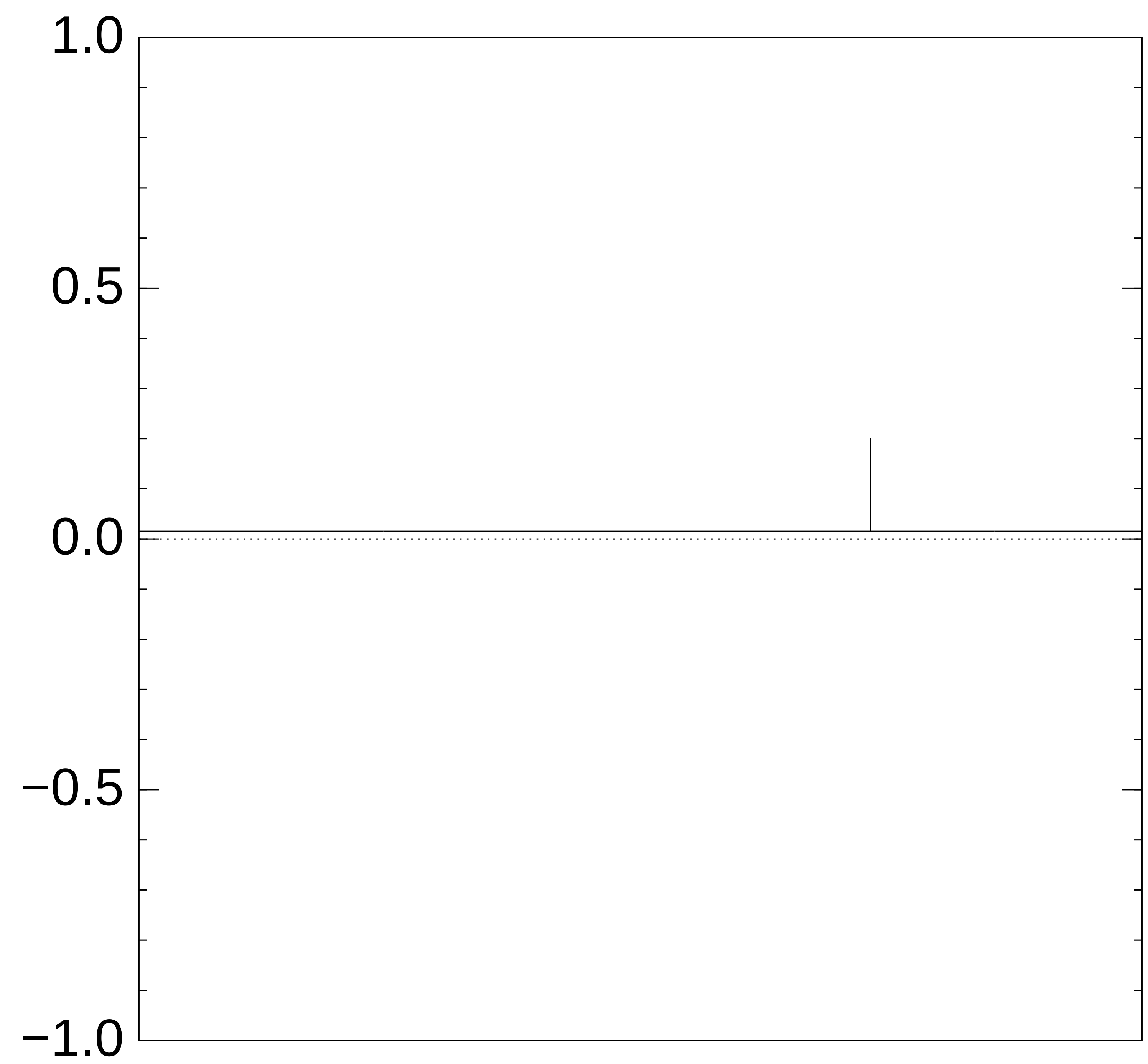
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $6 \times (\text{Step } 1 + \text{Step } 2)$:



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

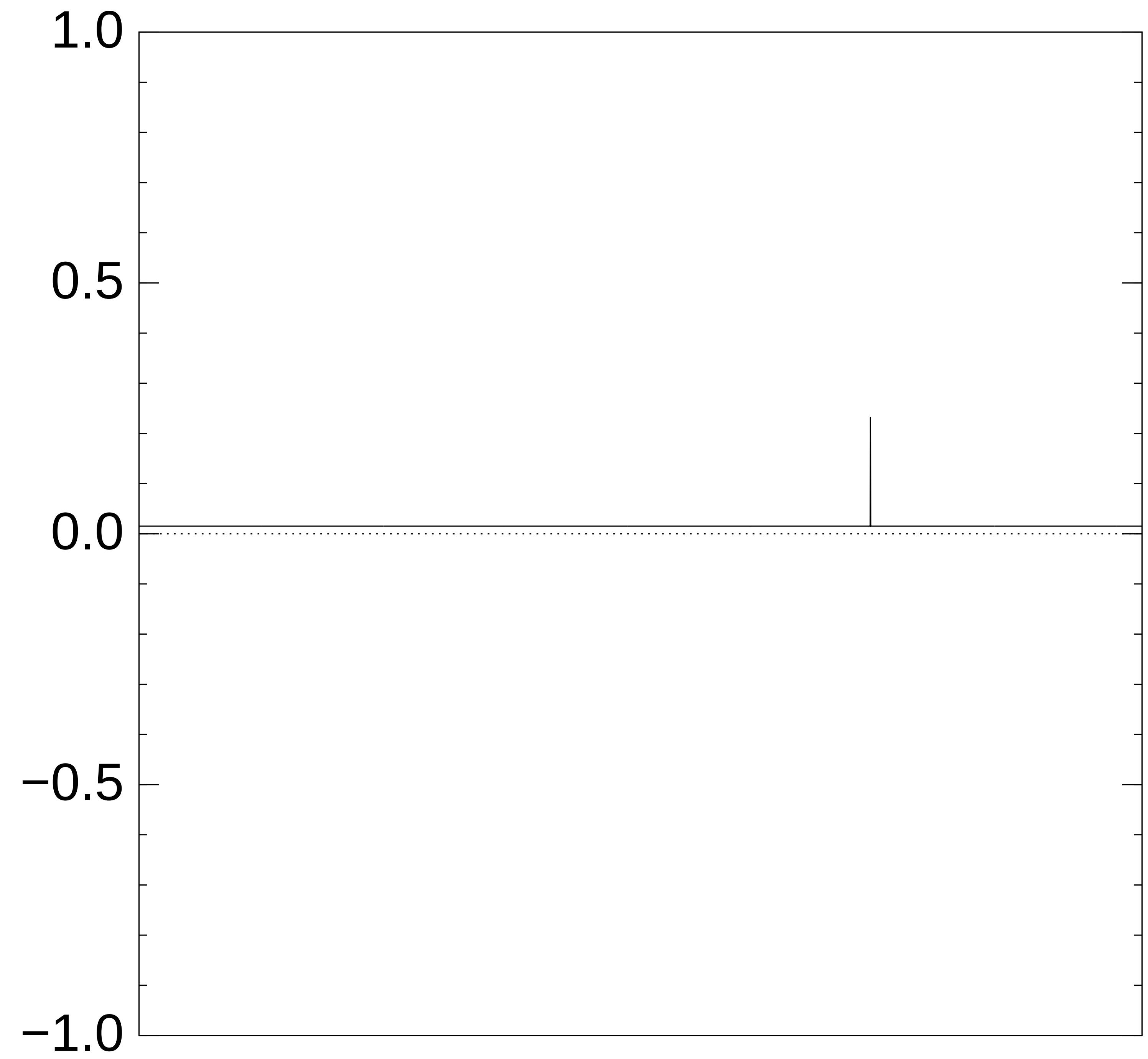
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $7 \times$ (Step 1 + Step 2):



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

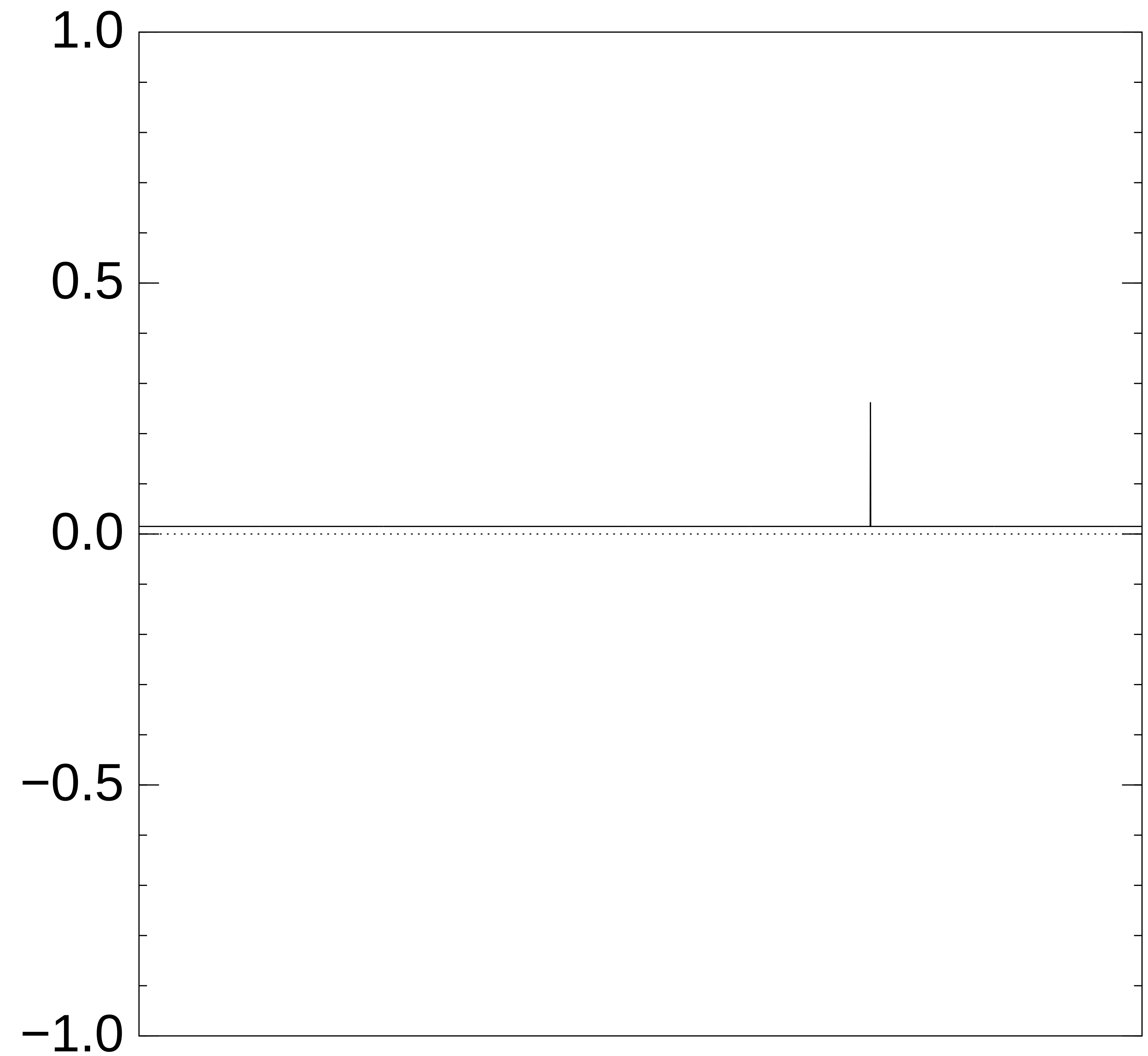
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $8 \times$ (Step 1 + Step 2):



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

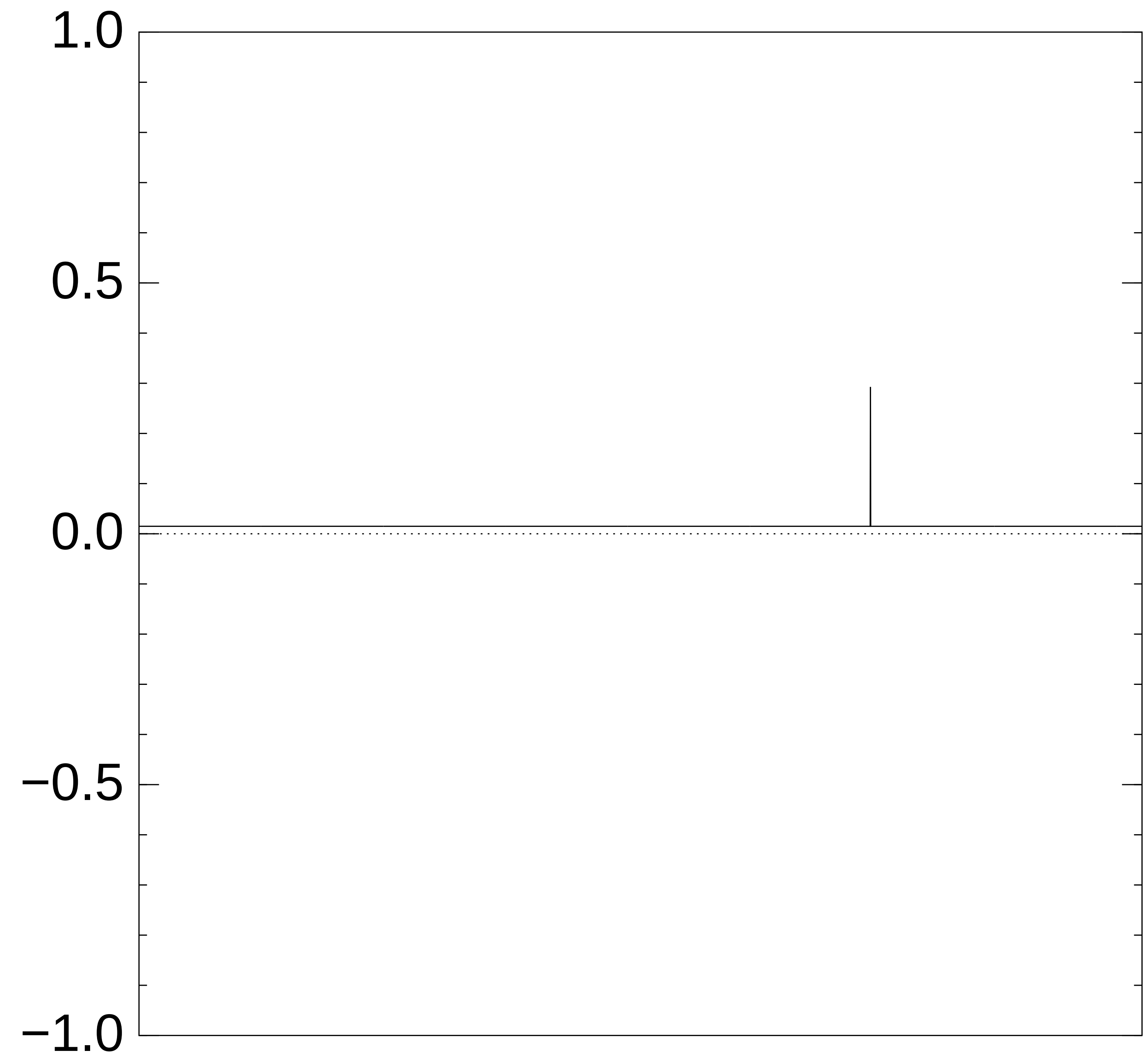
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $9 \times$ (Step 1 + Step 2):



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

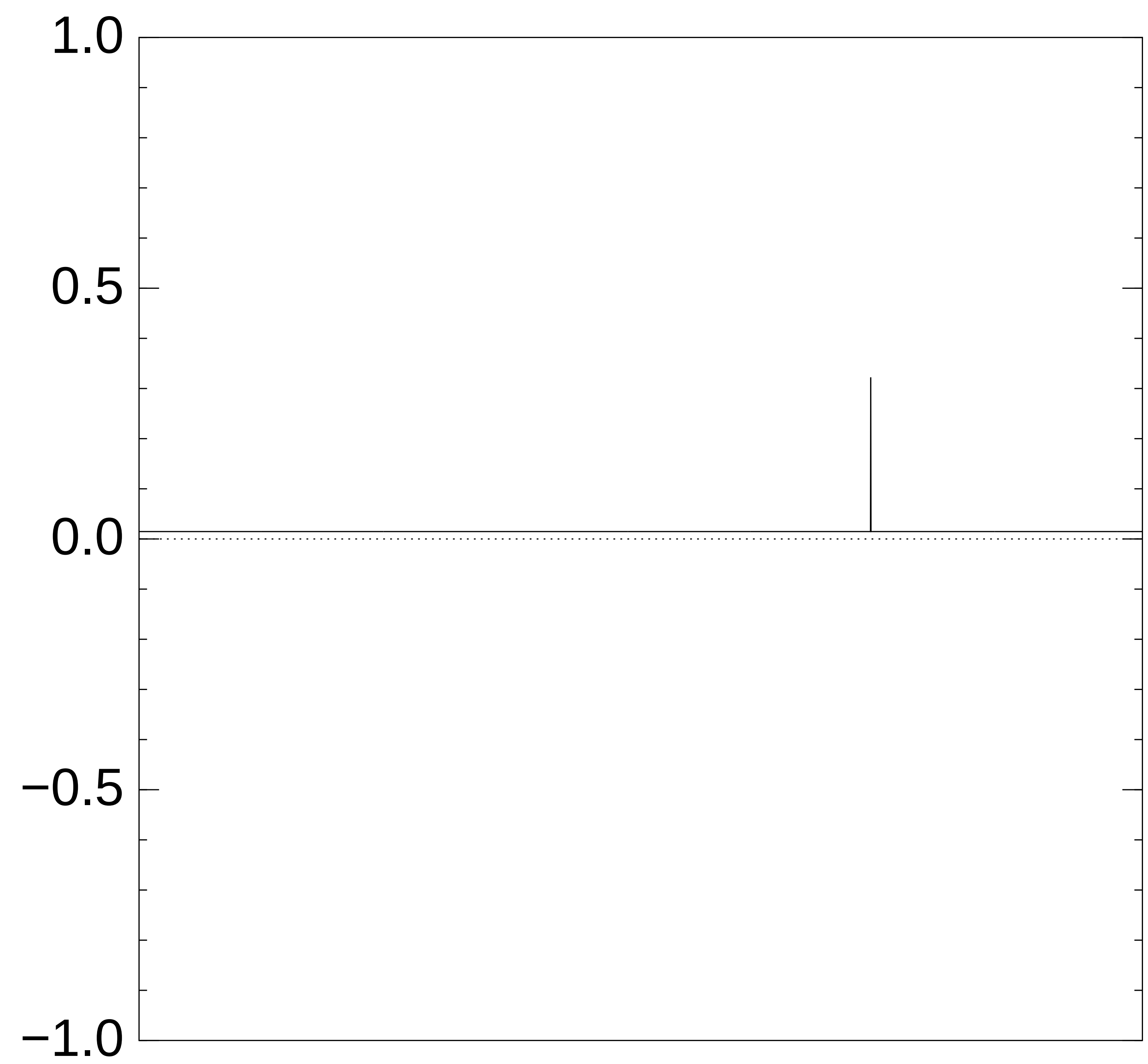
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $10 \times$ (Step 1 + Step 2):



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

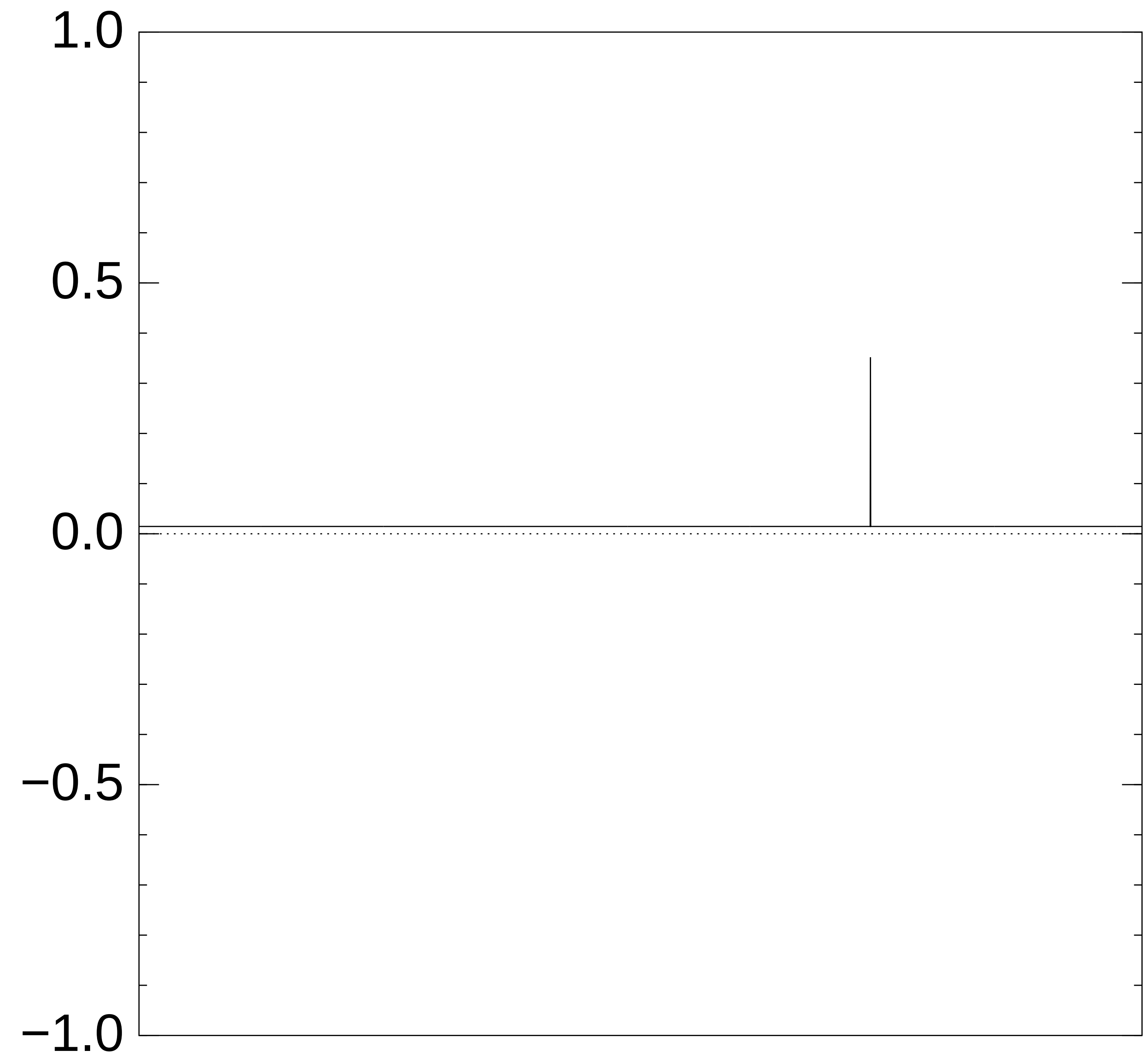
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $11 \times$ (Step 1 + Step 2):



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

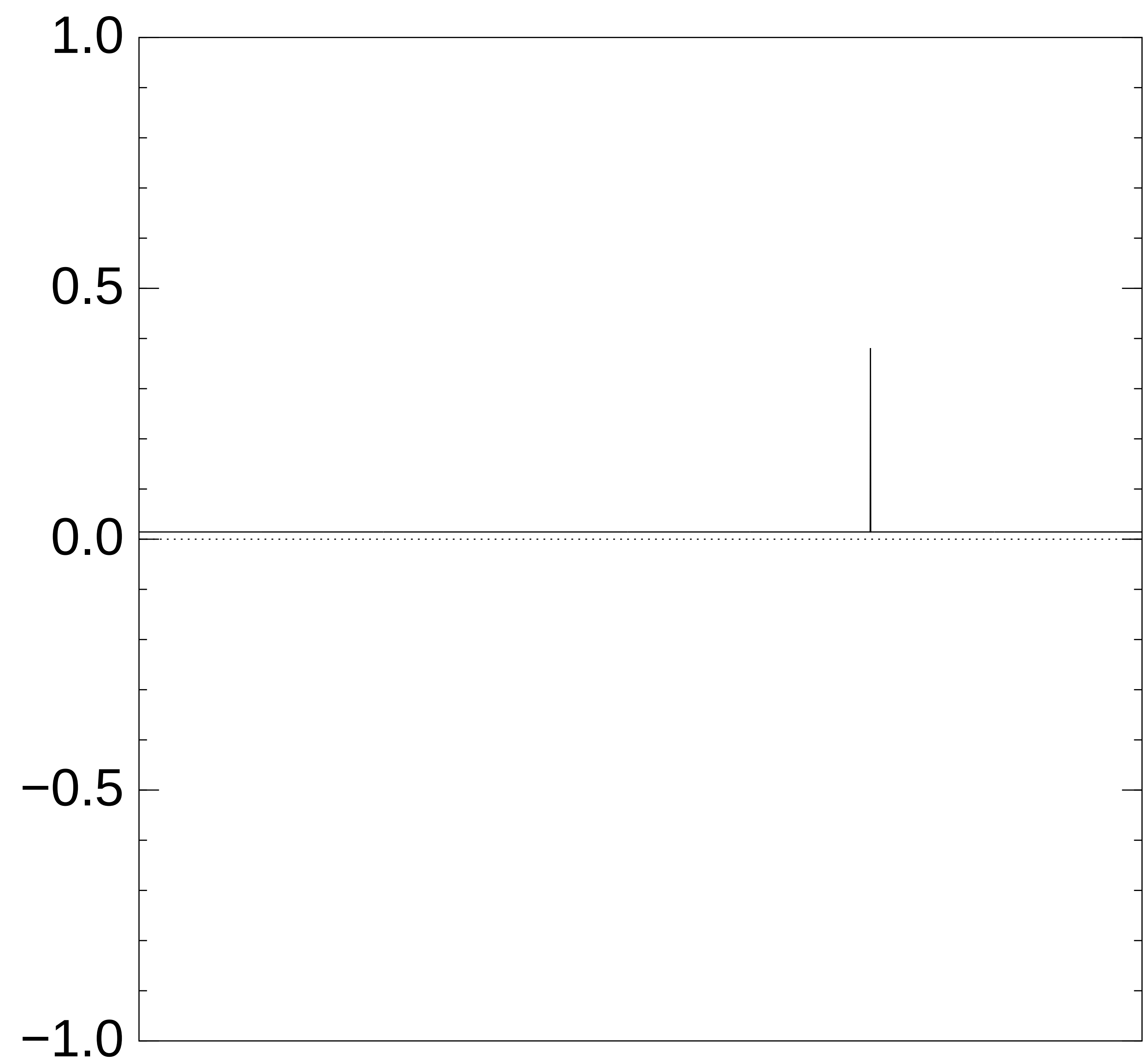
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $12 \times$ (Step 1 + Step 2):



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

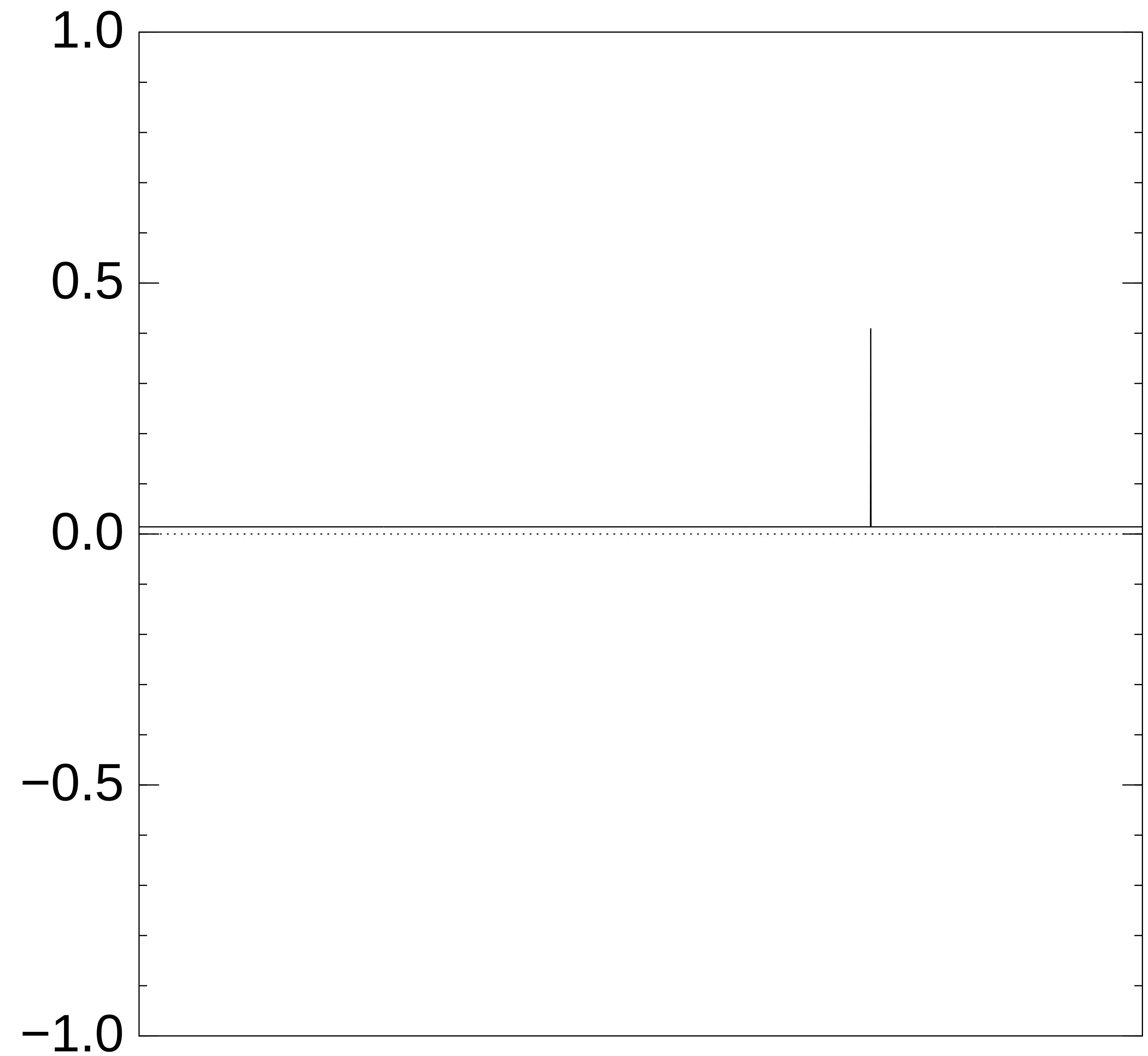
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $13 \times$ (Step 1 + Step 2):



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

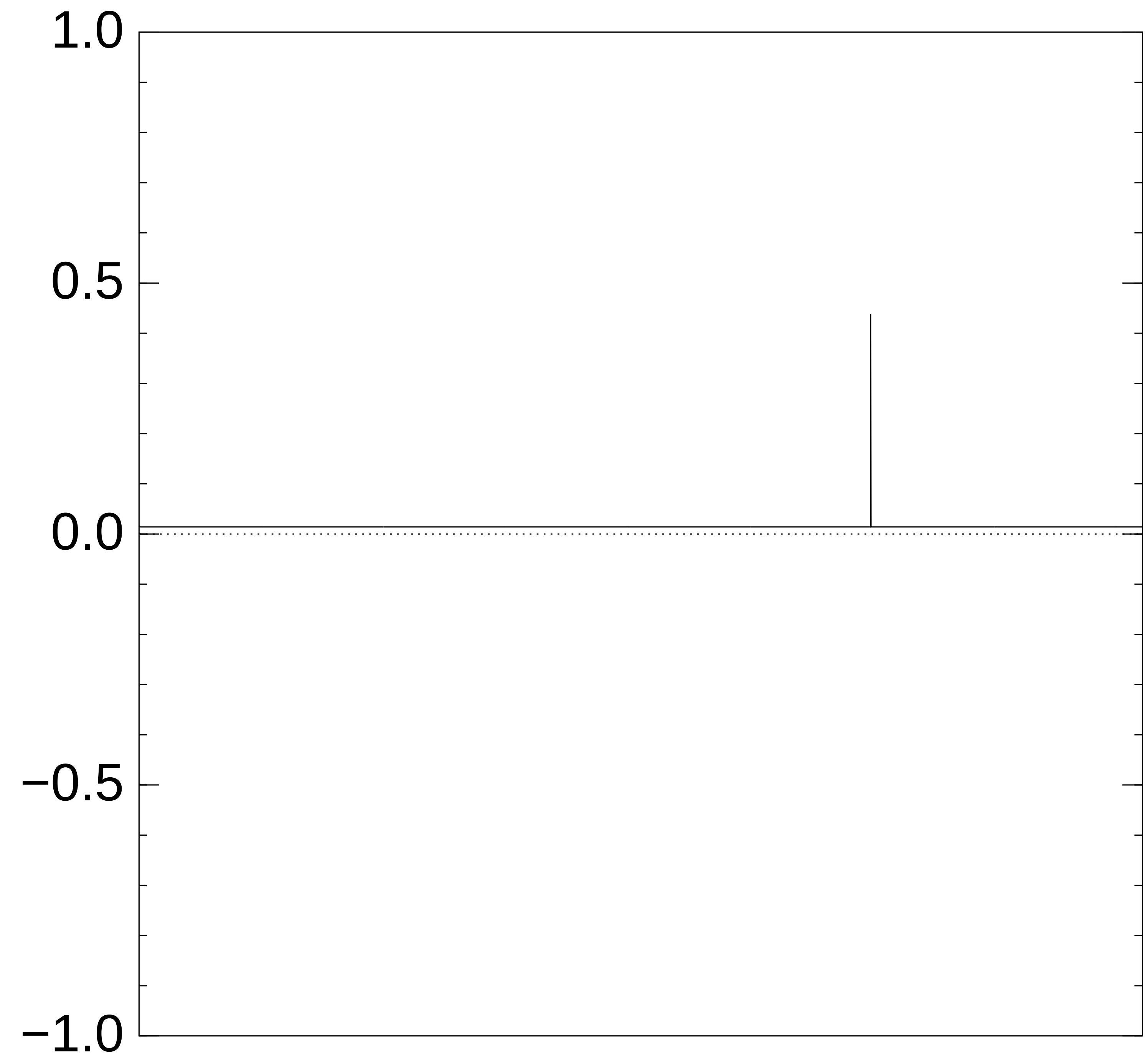
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $14 \times (\text{Step } 1 + \text{Step } 2)$:



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

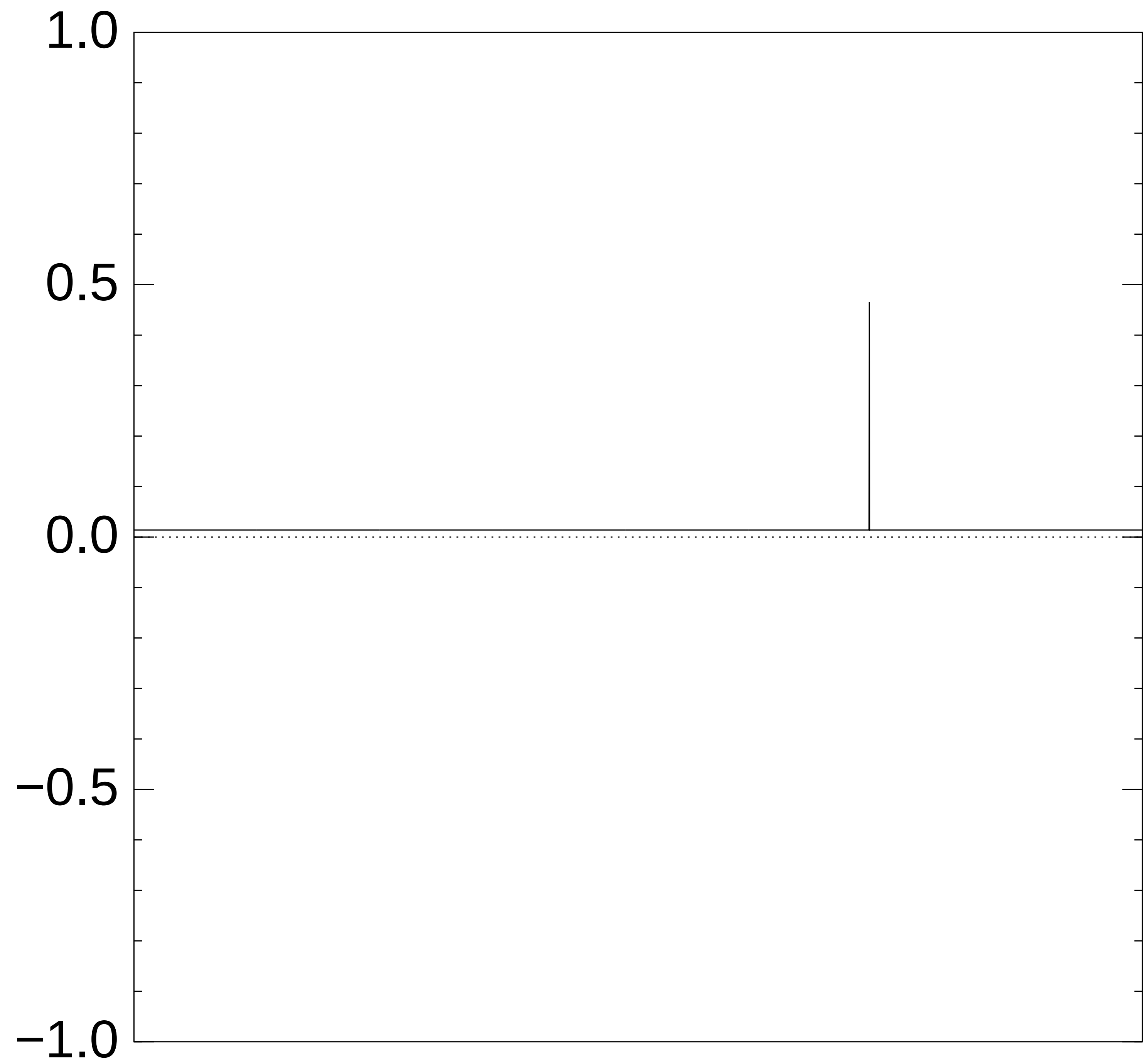
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $15 \times (\text{Step } 1 + \text{Step } 2)$:



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

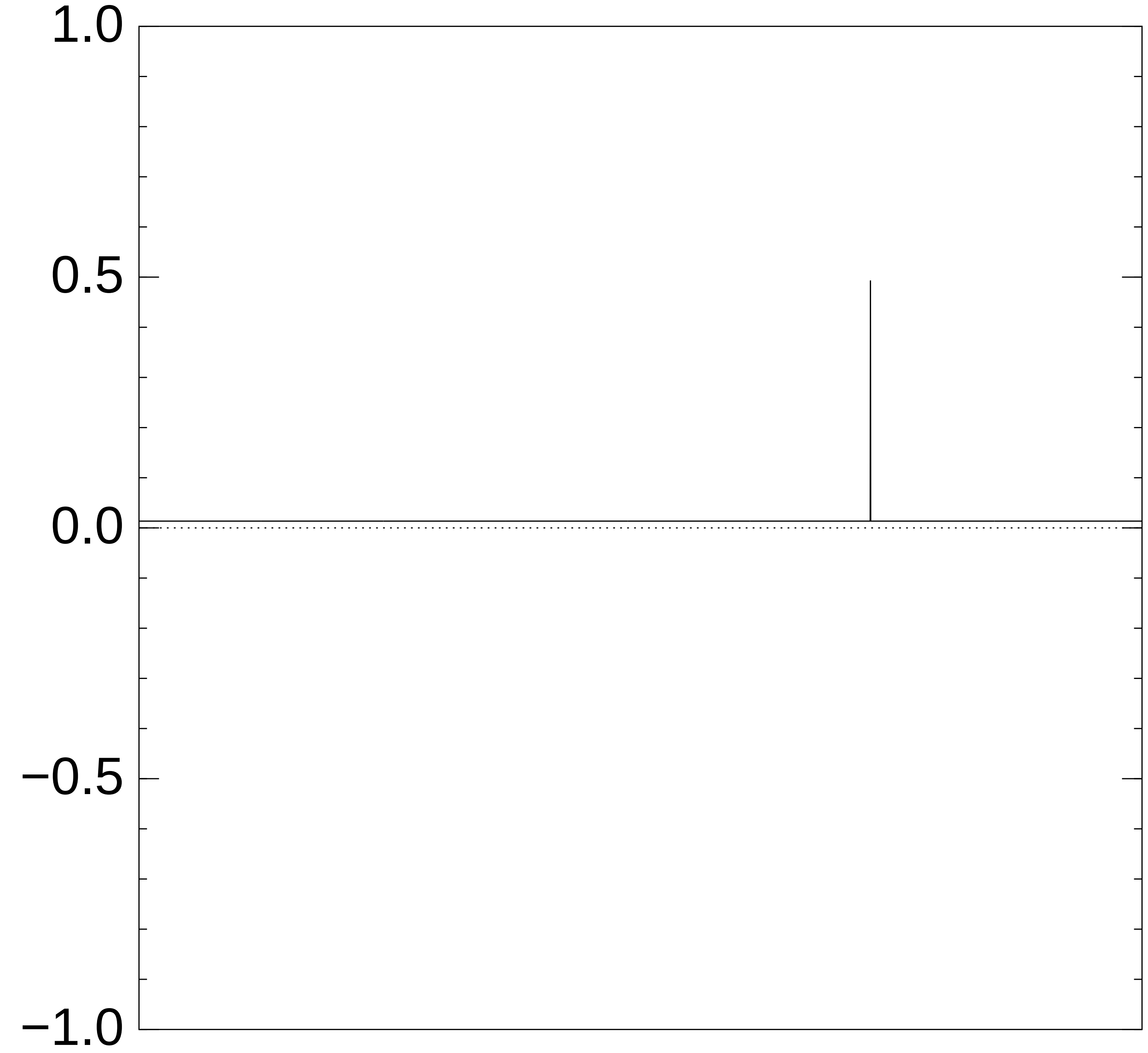
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $16 \times$ (Step 1 + Step 2):



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

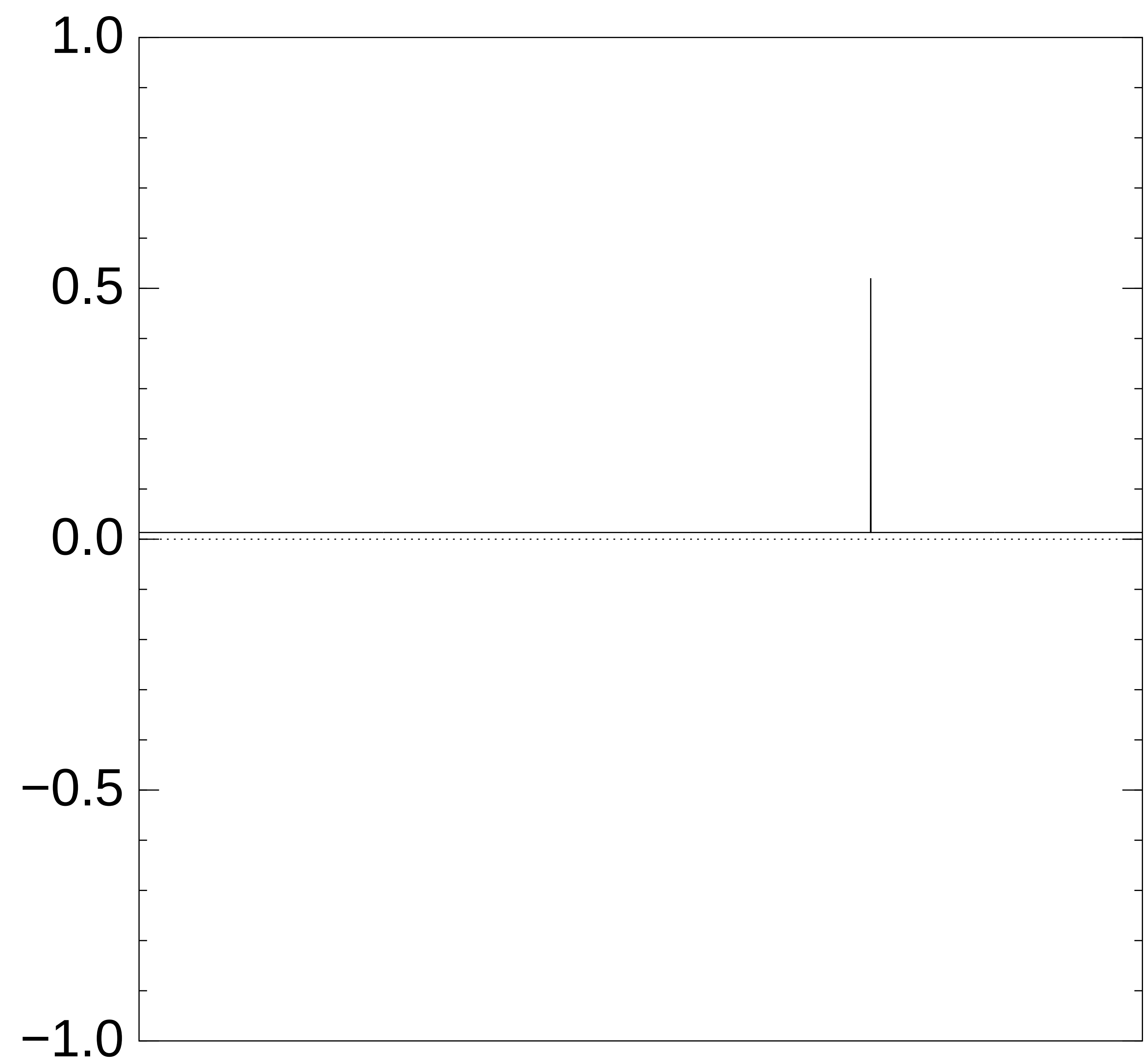
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $17 \times$ (Step 1 + Step 2):



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

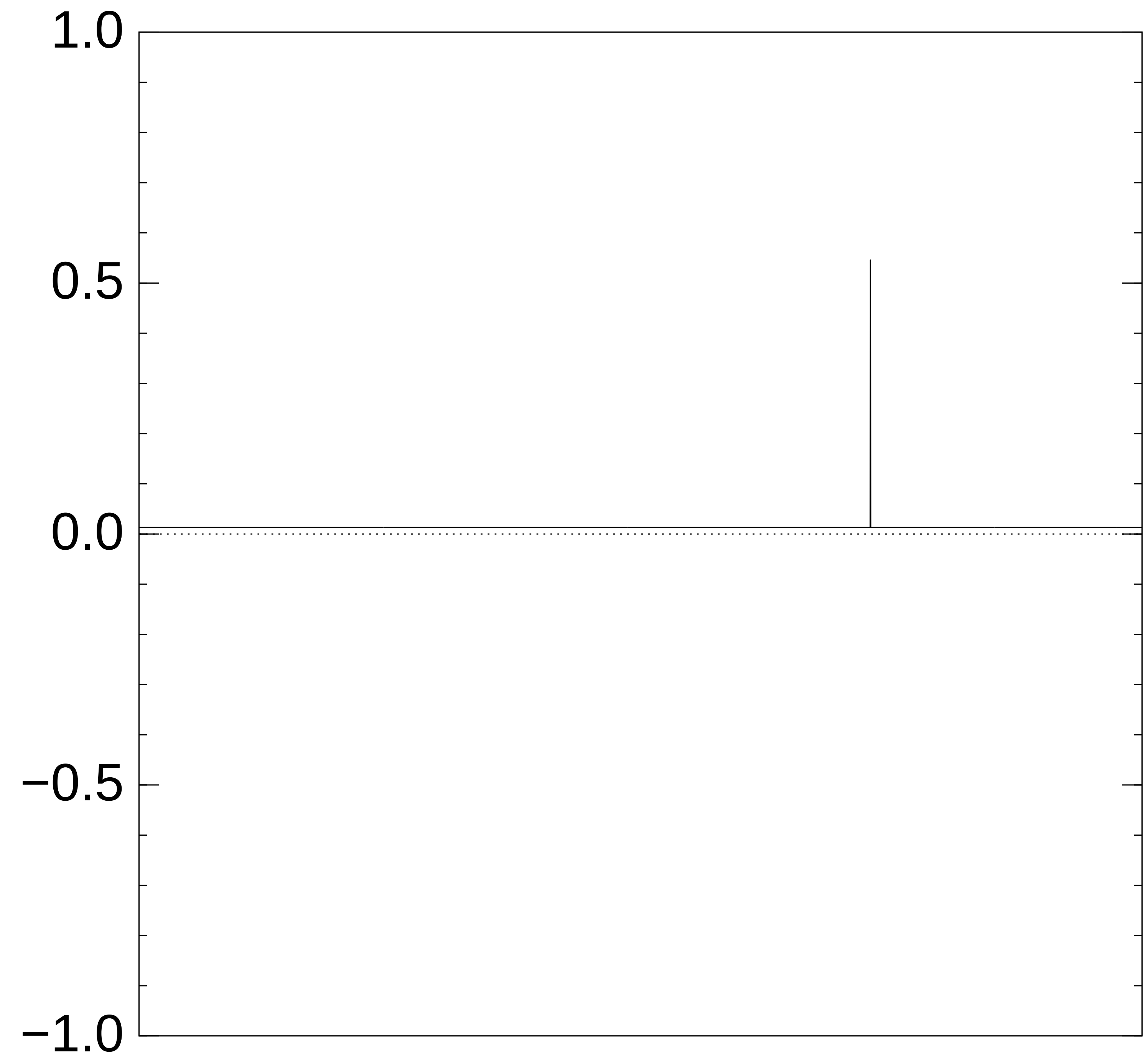
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $18 \times$ (Step 1 + Step 2):



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

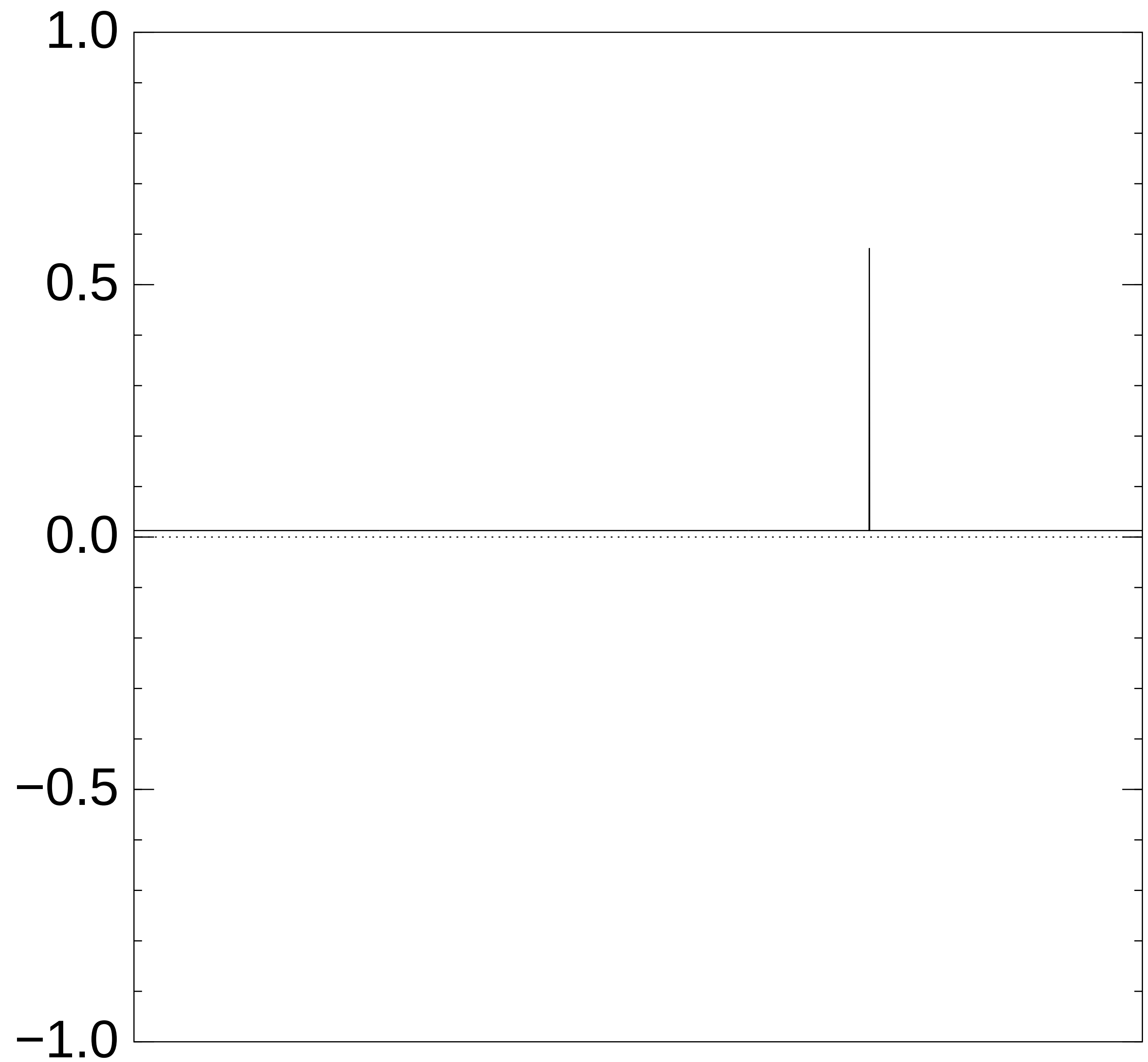
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $19 \times$ (Step 1 + Step 2):



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

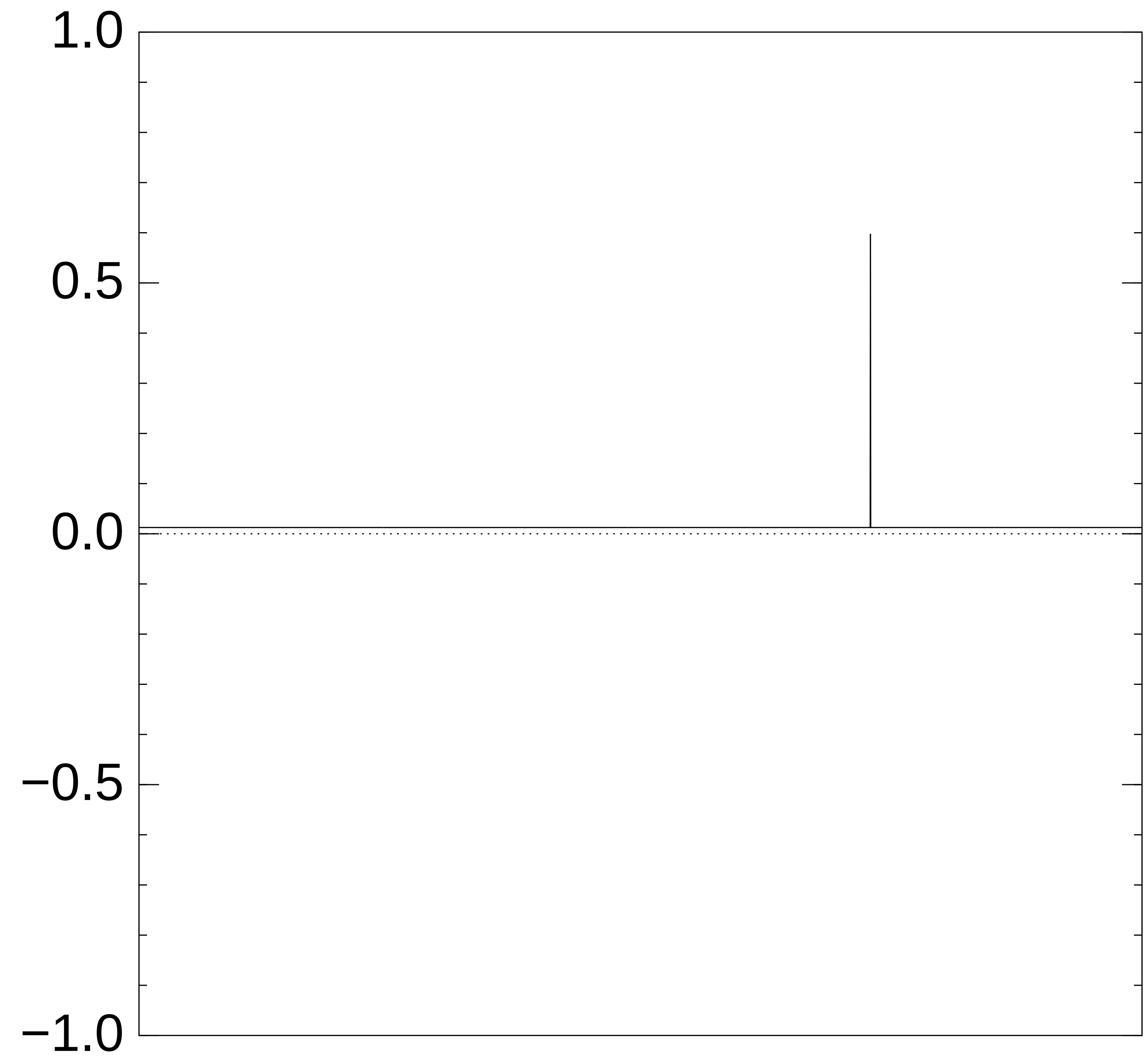
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $20 \times$ (Step 1 + Step 2):



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

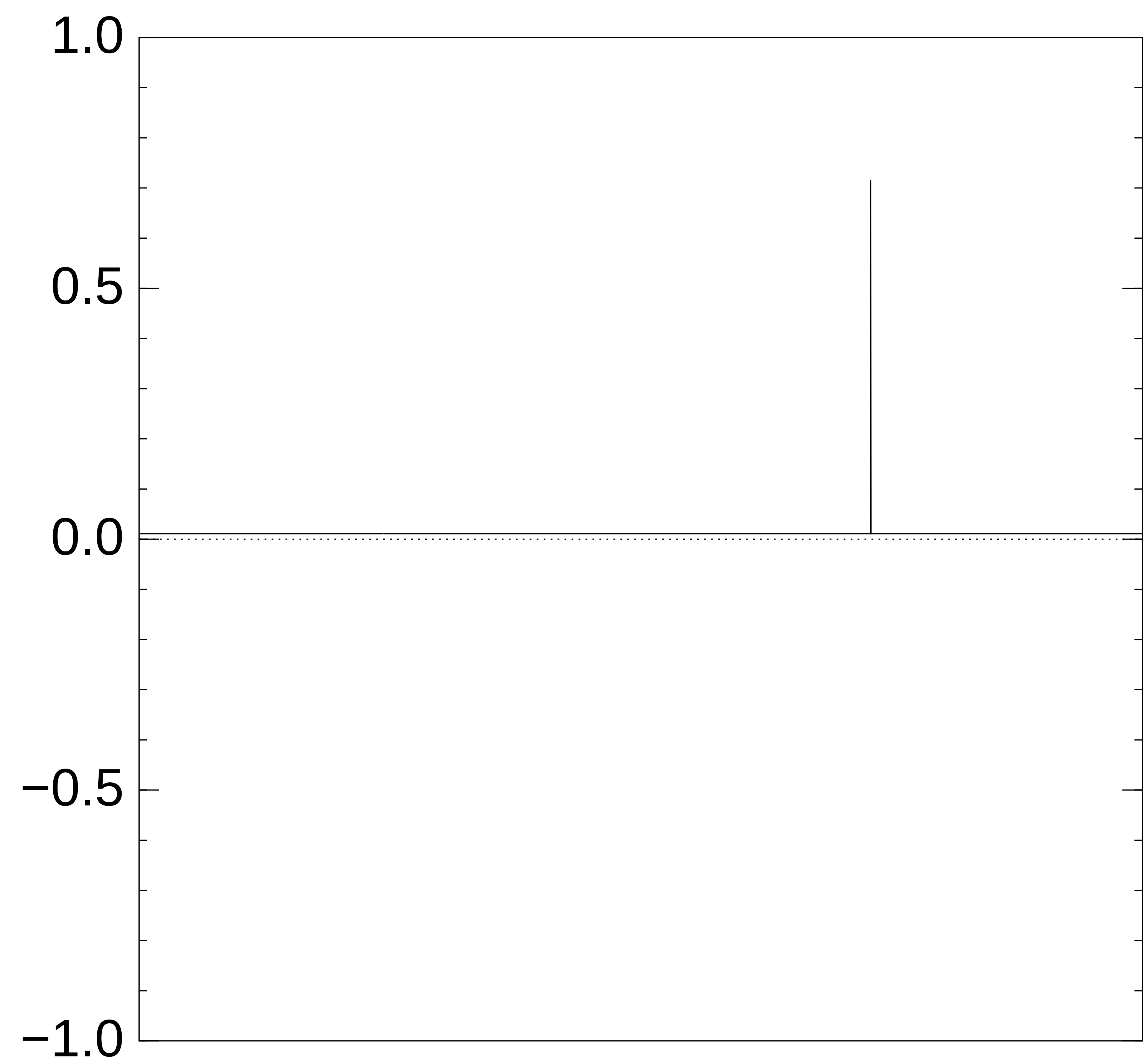
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $25 \times$ (Step 1 + Step 2):



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

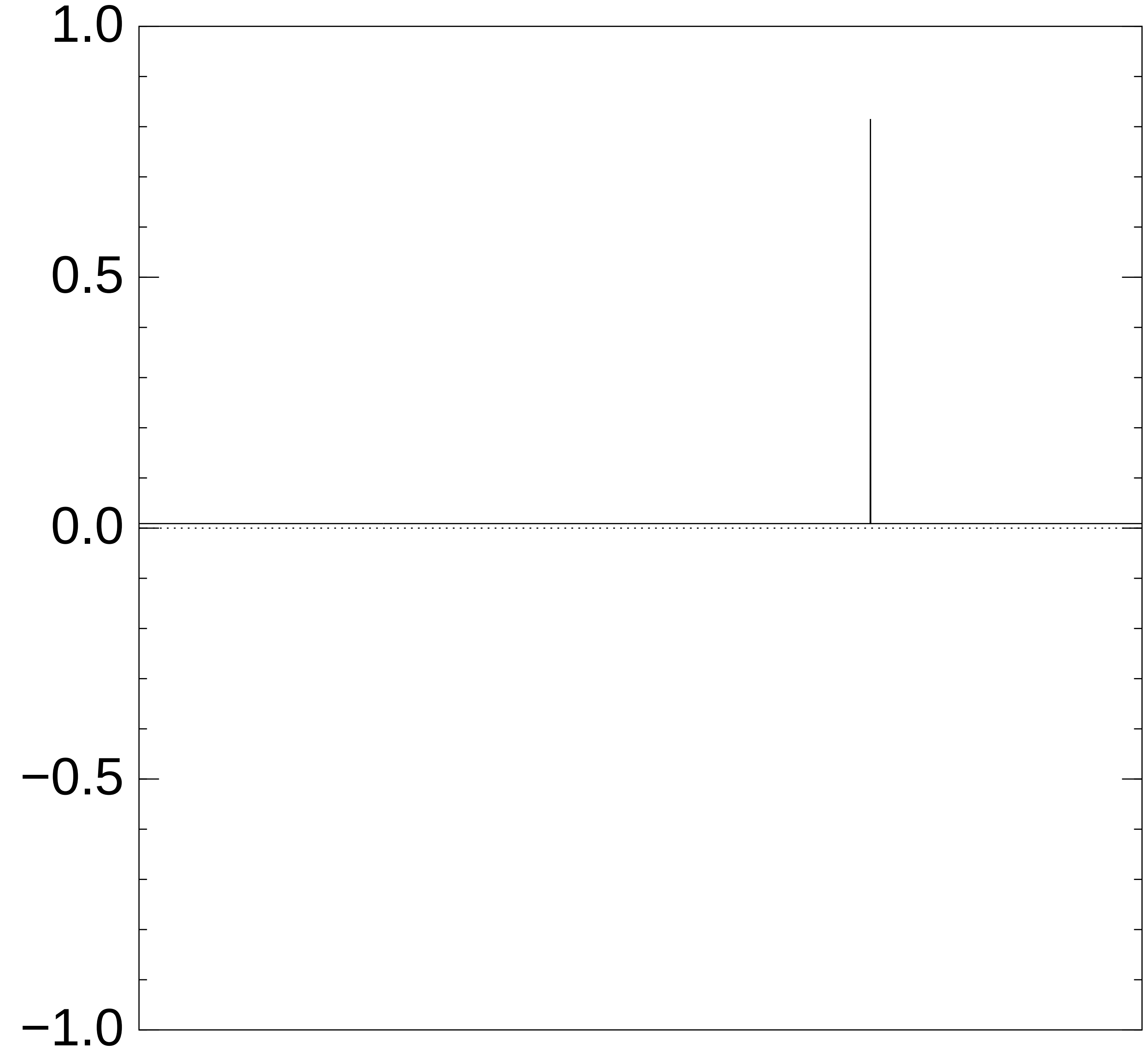
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $30 \times$ (Step 1 + Step 2):



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

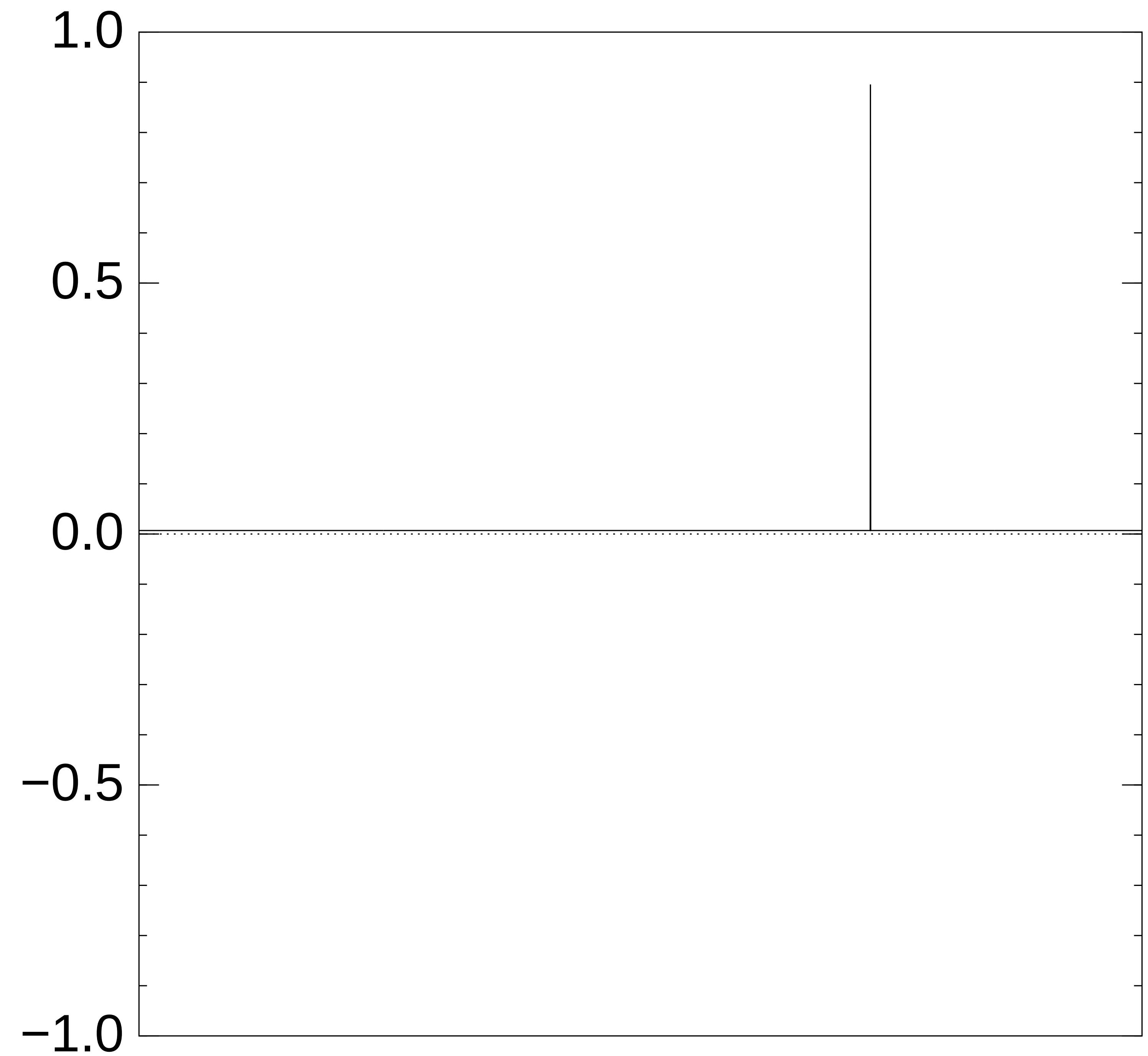
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ 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 u.

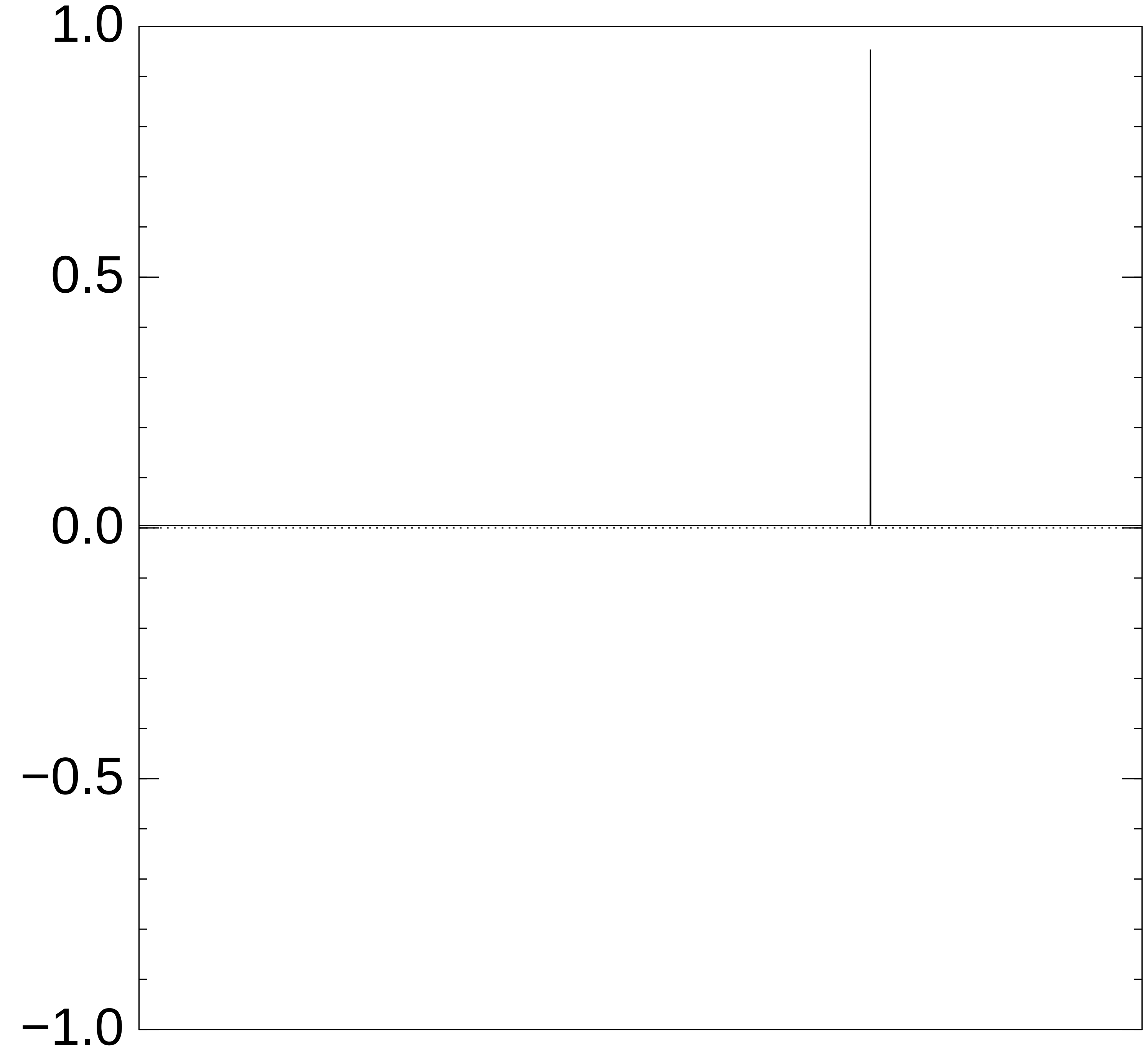
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $40 \times$ (Step 1 + Step 2):



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

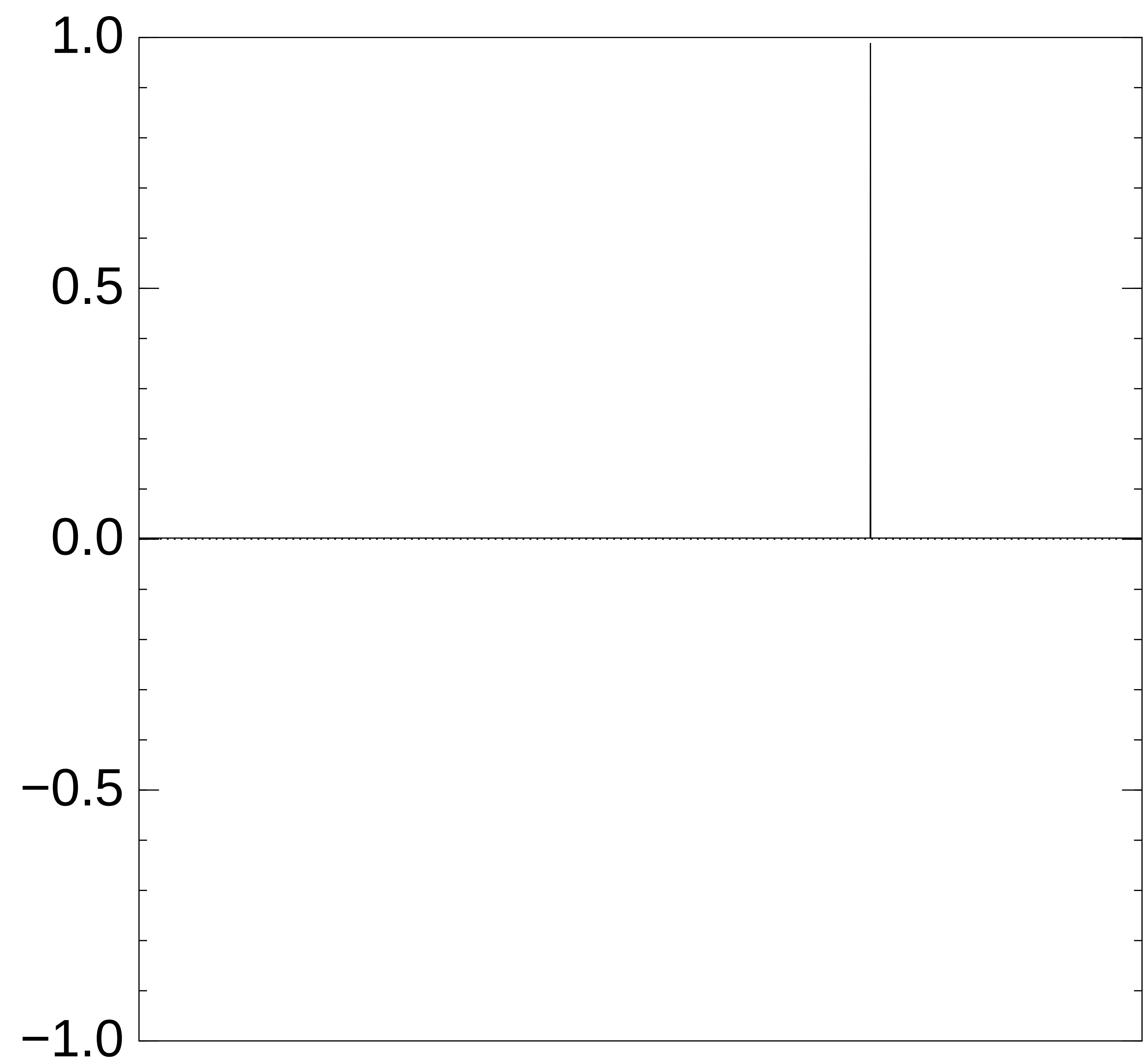
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $45 \times$ (Step 1 + Step 2):



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

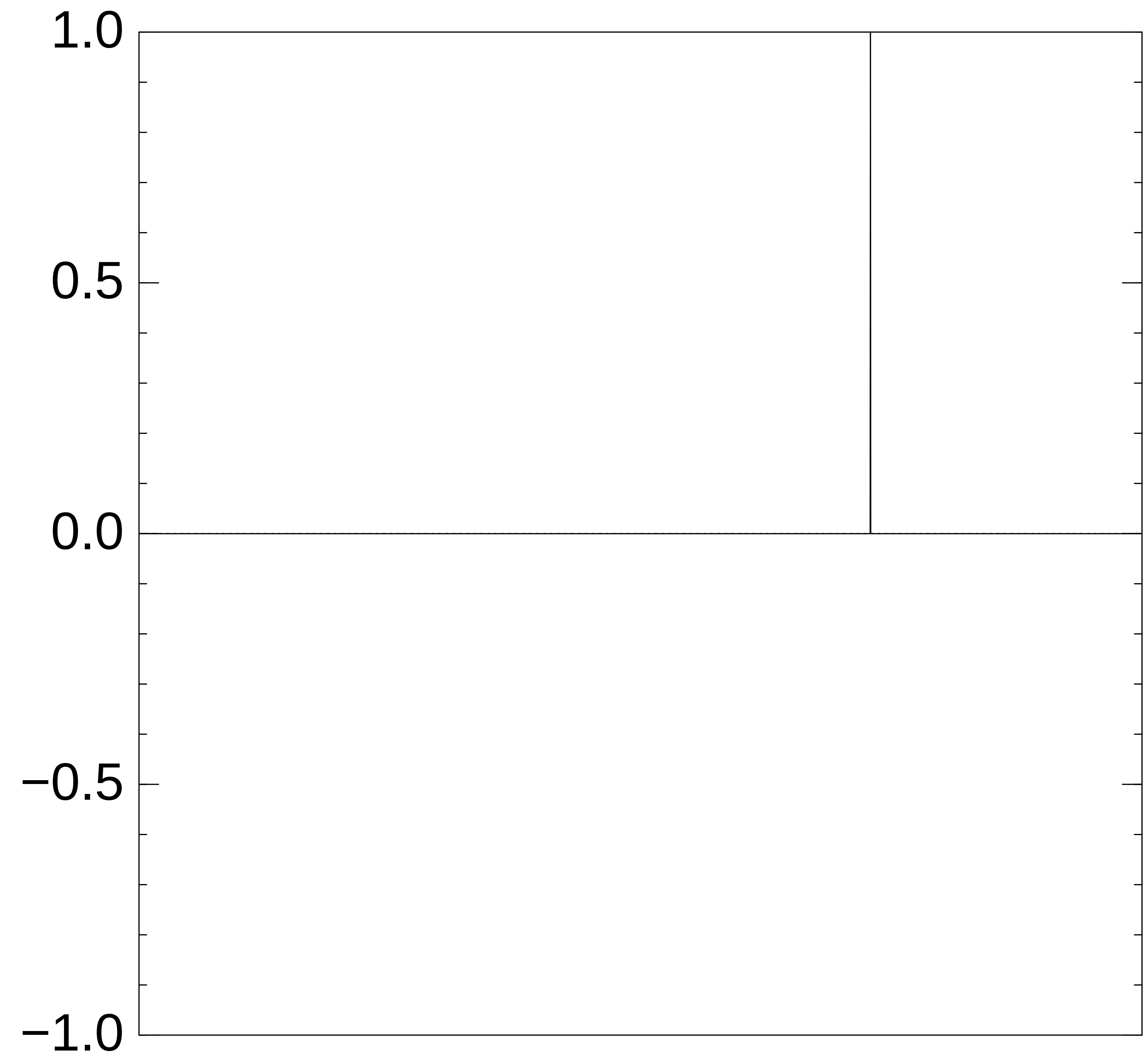
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ 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 u.

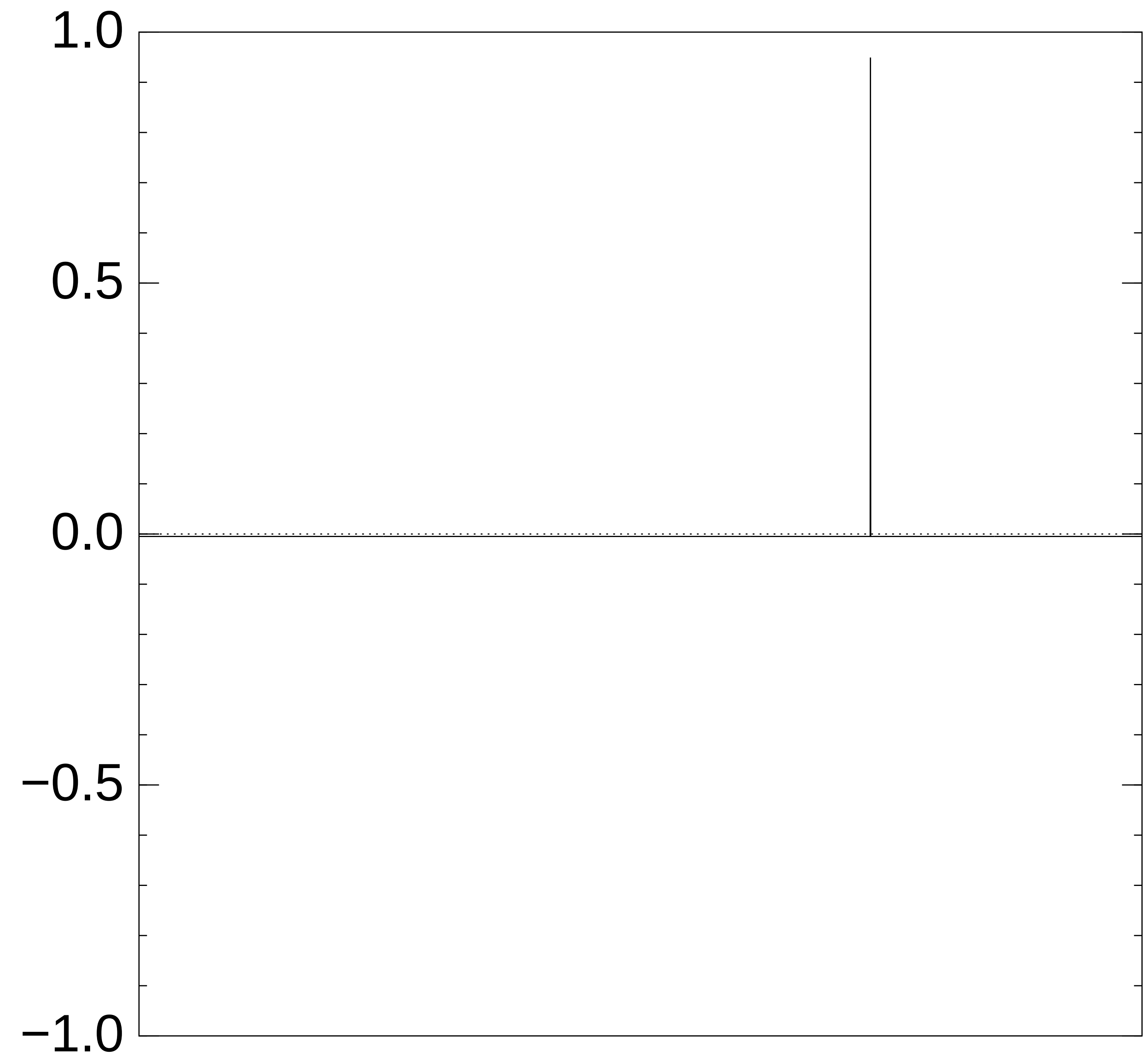
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $60 \times (\text{Step } 1 + \text{Step } 2)$:



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

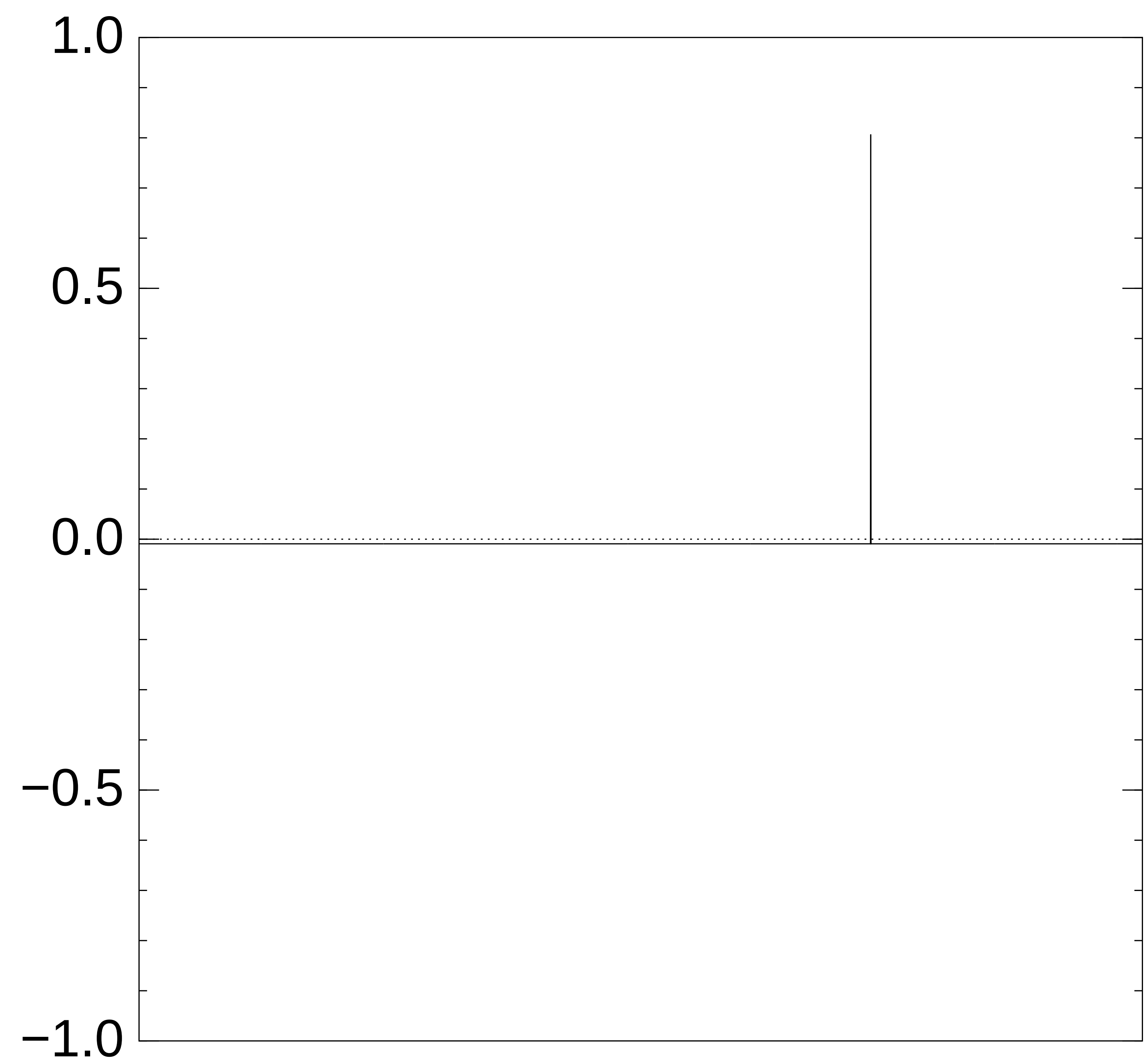
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $70 \times$ (Step 1 + Step 2):



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

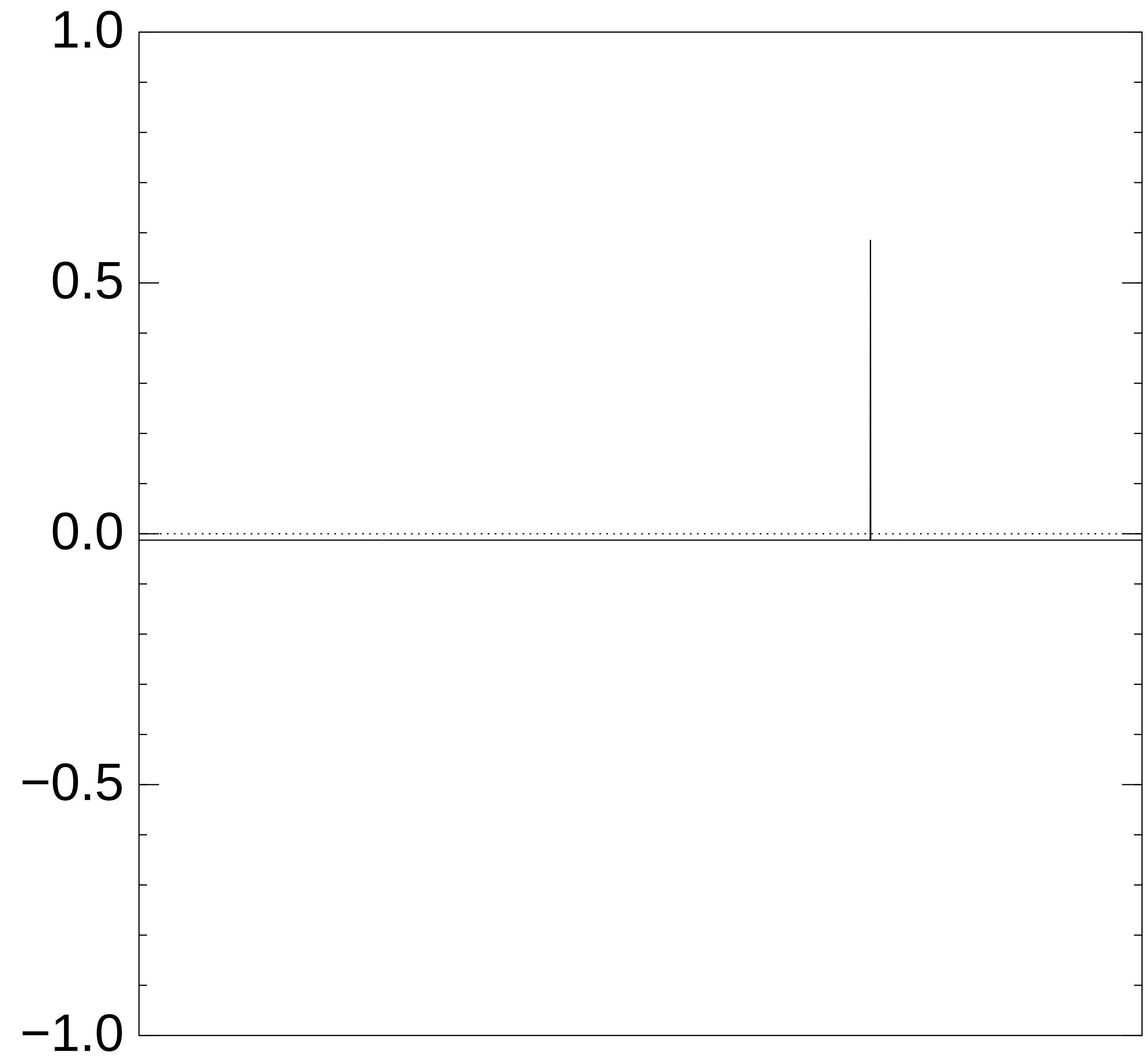
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $80 \times$ (Step 1 + Step 2):



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

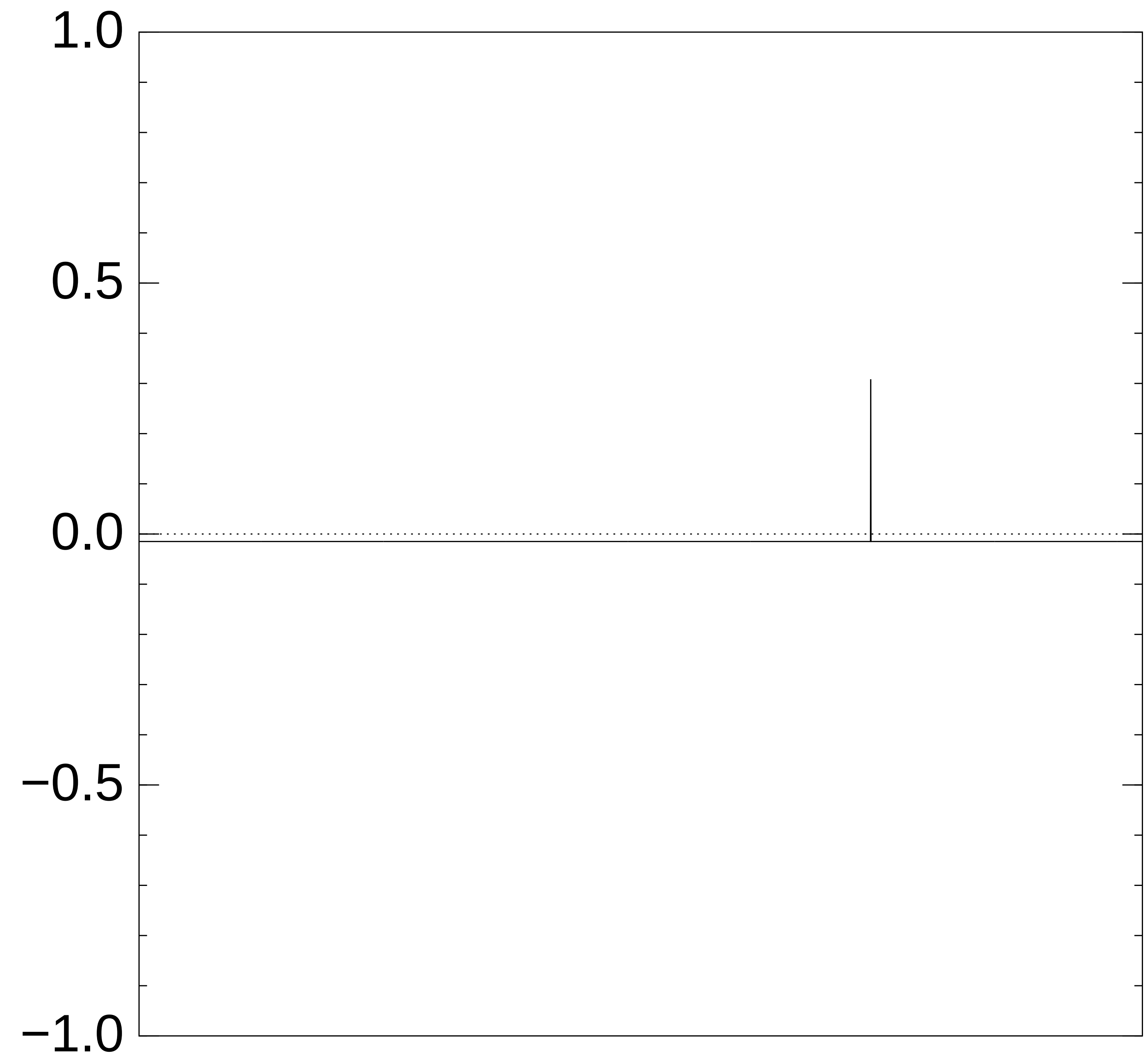
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $90 \times$ (Step 1 + Step 2):



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

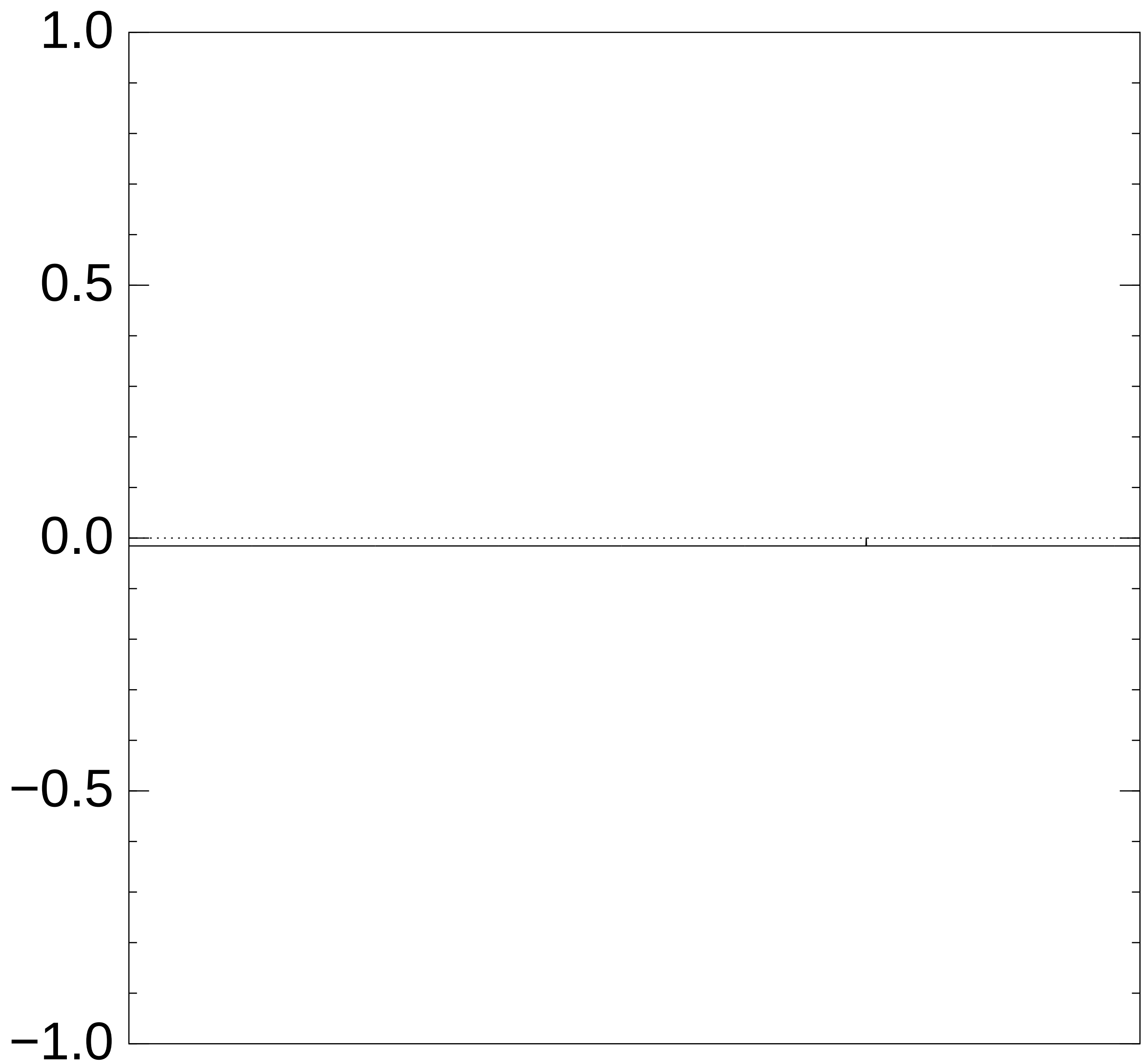
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast.

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 $u \mapsto a_u$ for an example with $n = 12$ after $100 \times$ (Step 1 + Step 2):



Very bad stopping point.

om uniform superposition
n-bit strings u.

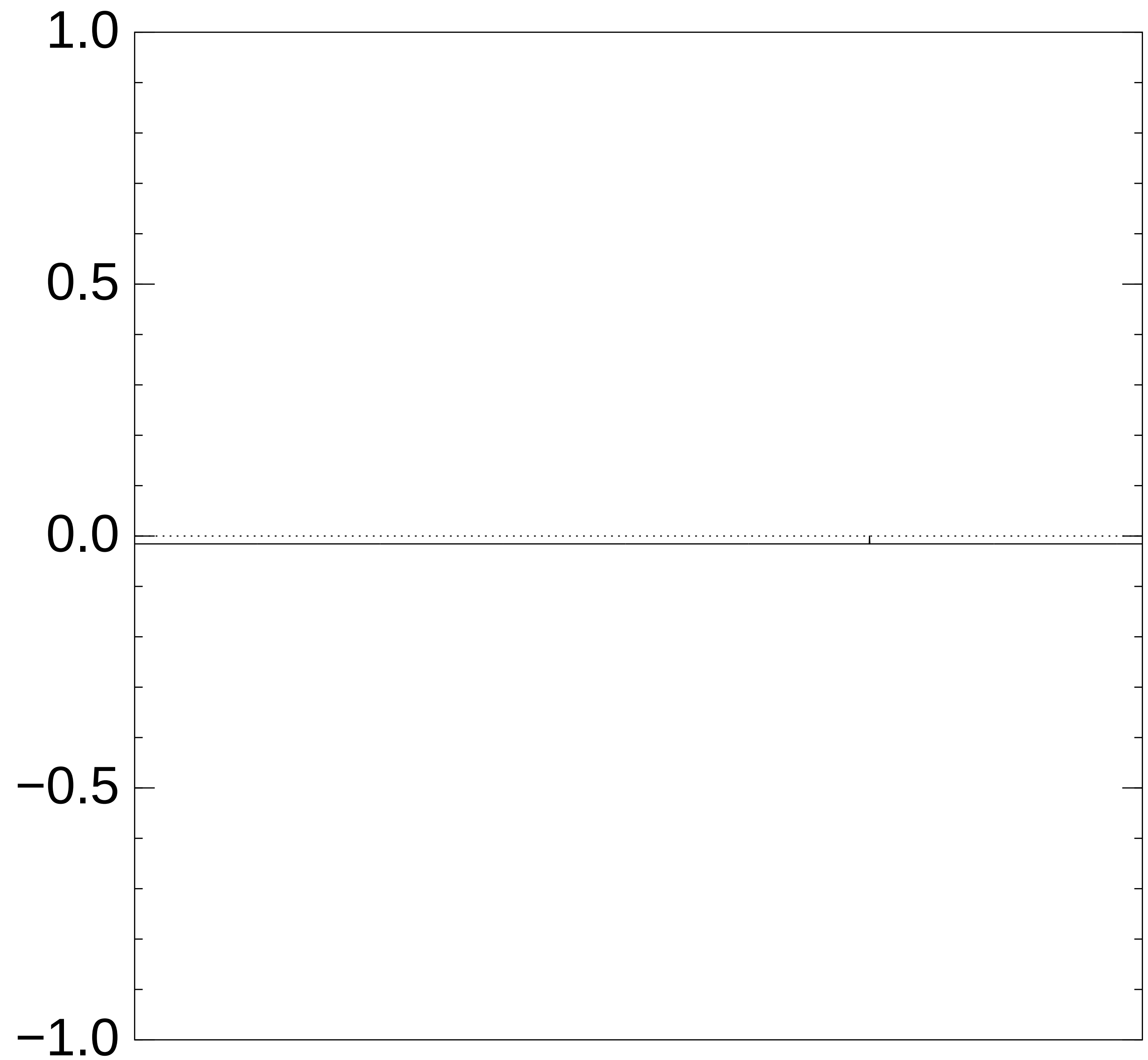
Set $a \leftarrow b$ where
a_u if $f(u) = 0$,
otherwise.
ast.

“Grover diffusion”.
a around its average.
lso fast.

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

the n qubits.
gh probability this finds s.

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



Very bad stopping point.

$u \mapsto a_u$
by a vec
(with fix
(1) a_u fo
(2) a_u fo

n superposition
gs u.

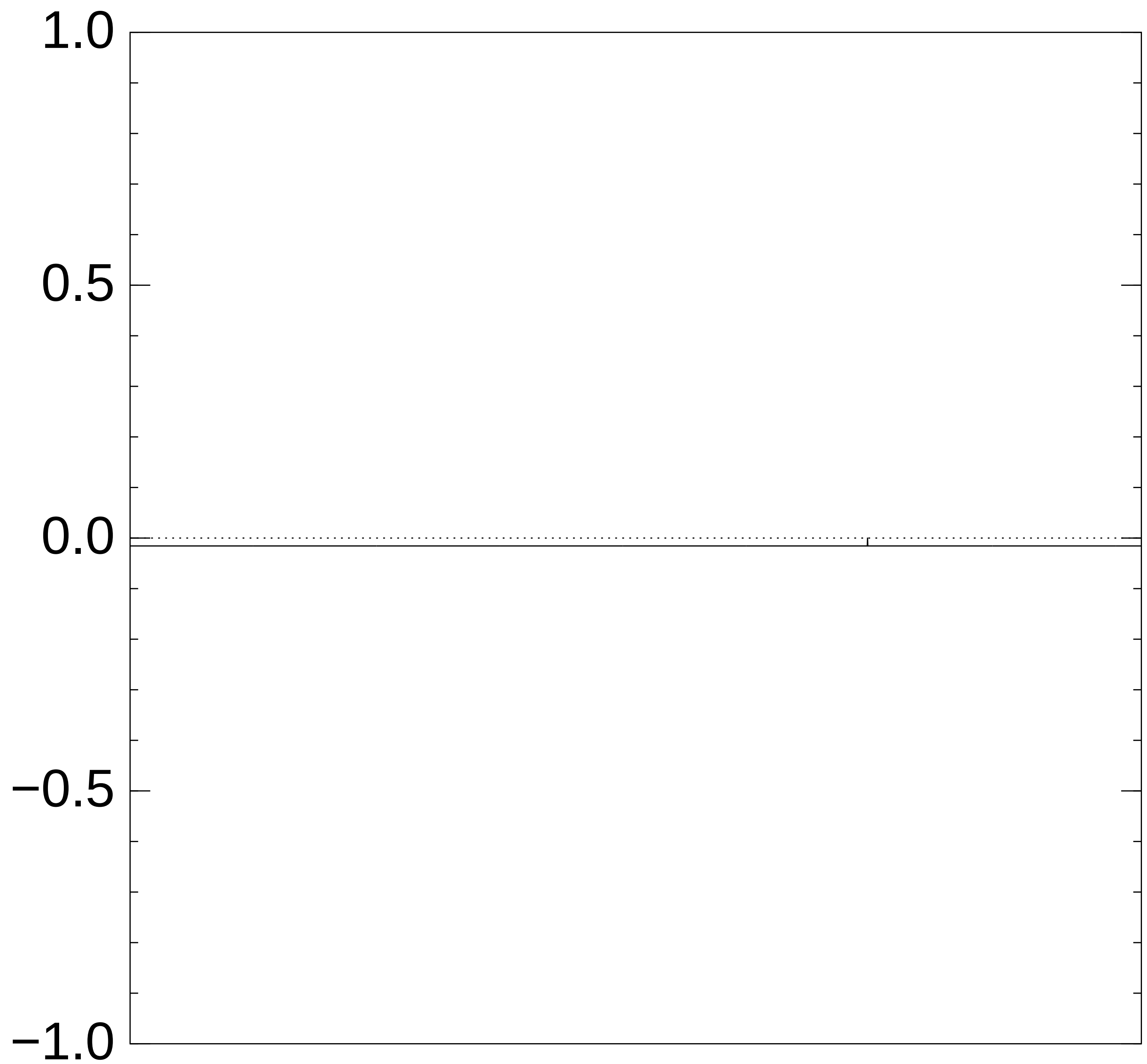
o where
$= 0$,
.

iffusion”.
ts average.

Step 2
times.

bits.
lity this finds s.

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



Very bad stopping point.

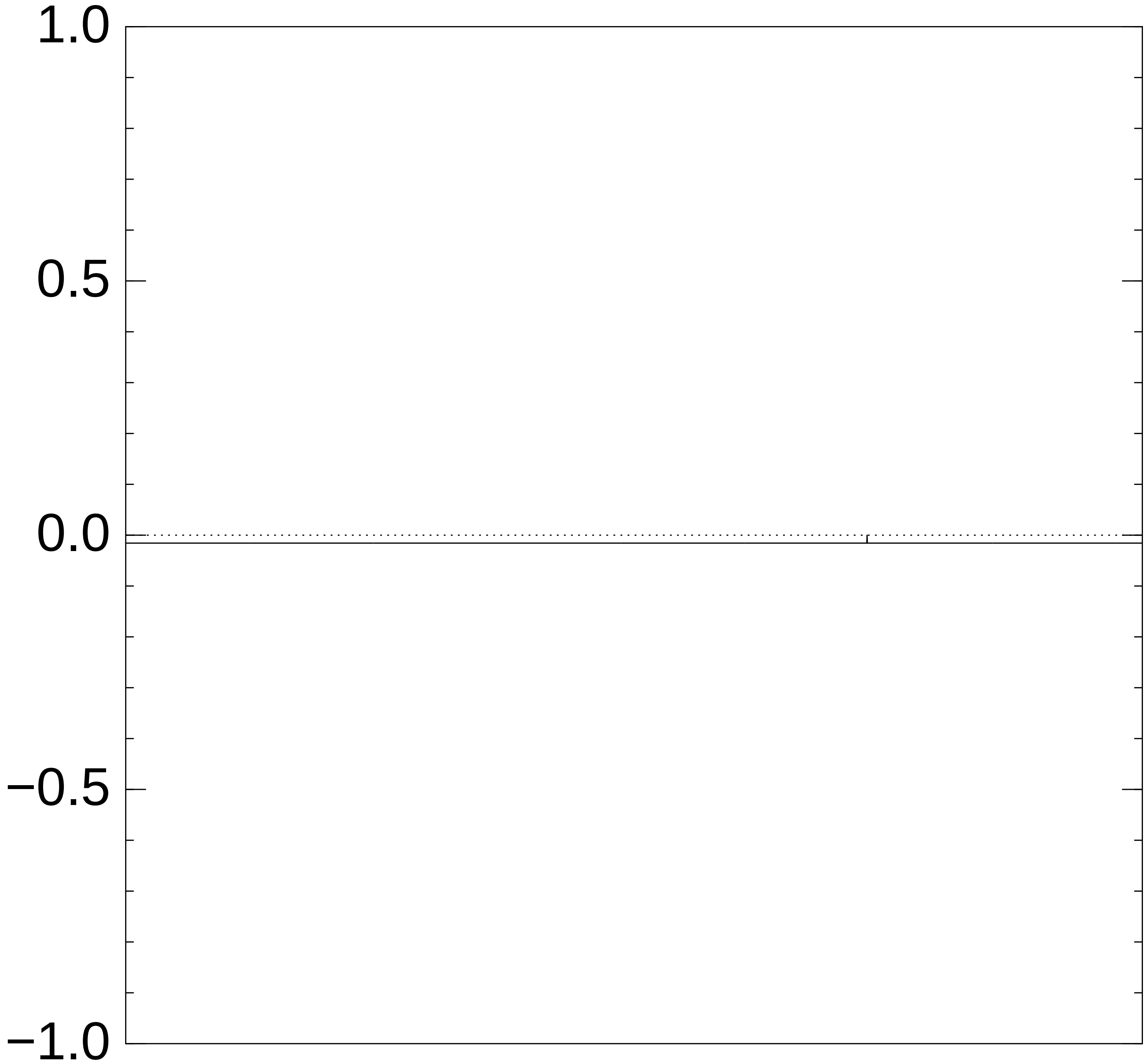
$u \mapsto a_u$ is complet
by a vector of two
(with fixed multipl
(1) a_u for roots u;
(2) a_u for non-roo

sition

.

nds s.

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



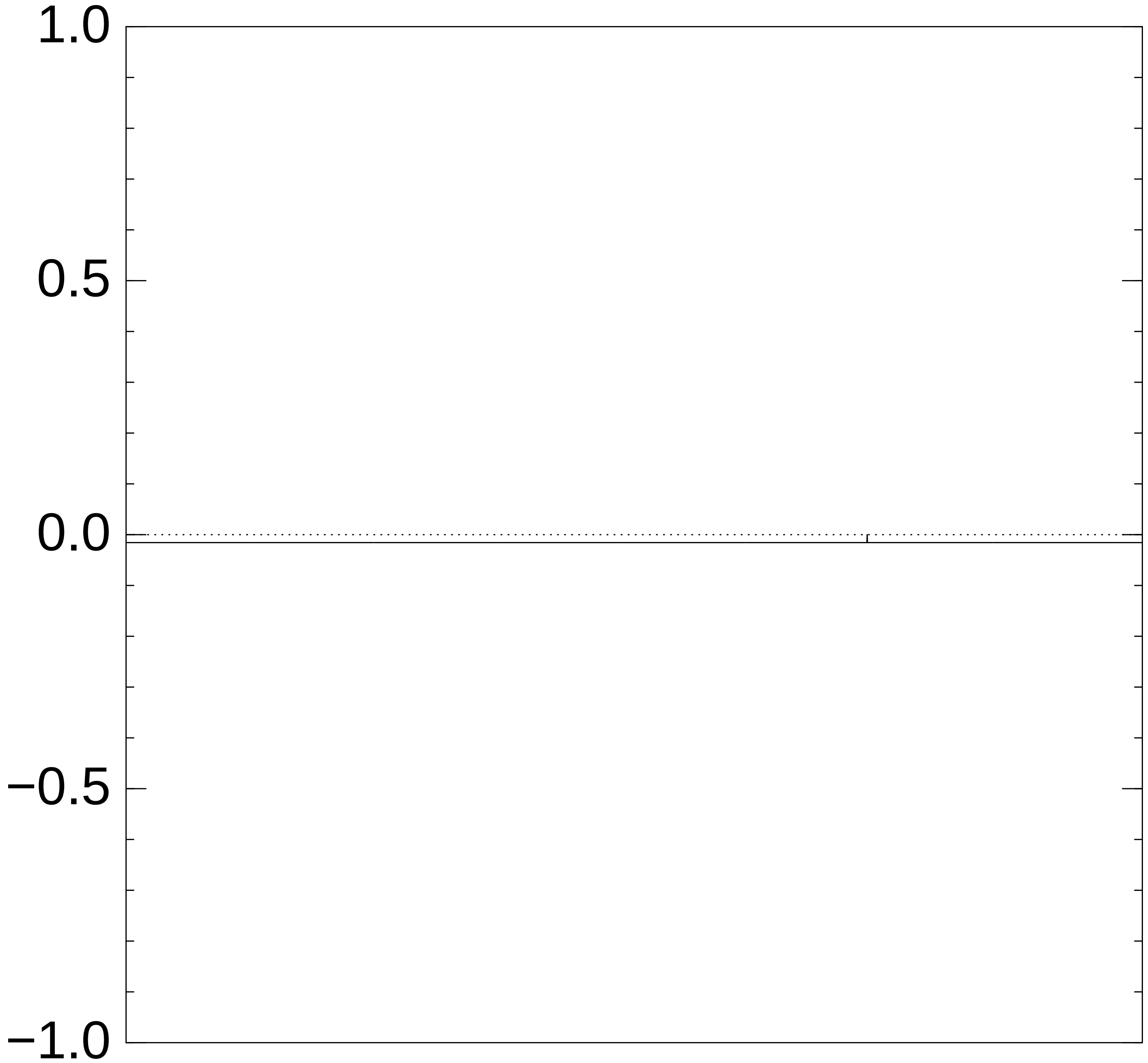
Very bad stopping point.

$u \mapsto a_u$ is completely describ by a vector of two numbers (with fixed multiplicities):

(1) a_u for roots u;

(2) a_u for non-roots u.

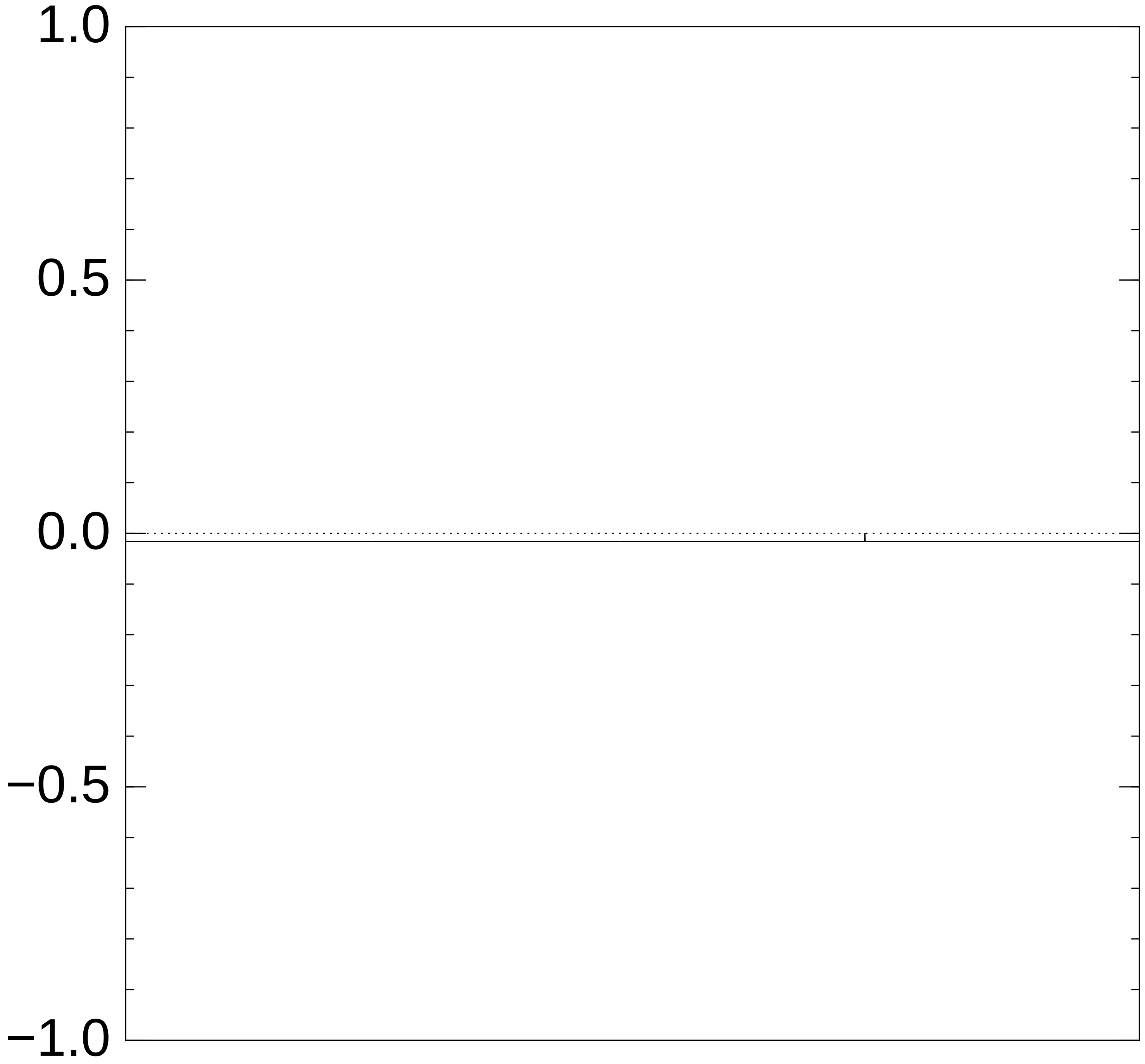
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $100 \times$ (Step 1 + Step 2):



Very bad stopping point.

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

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

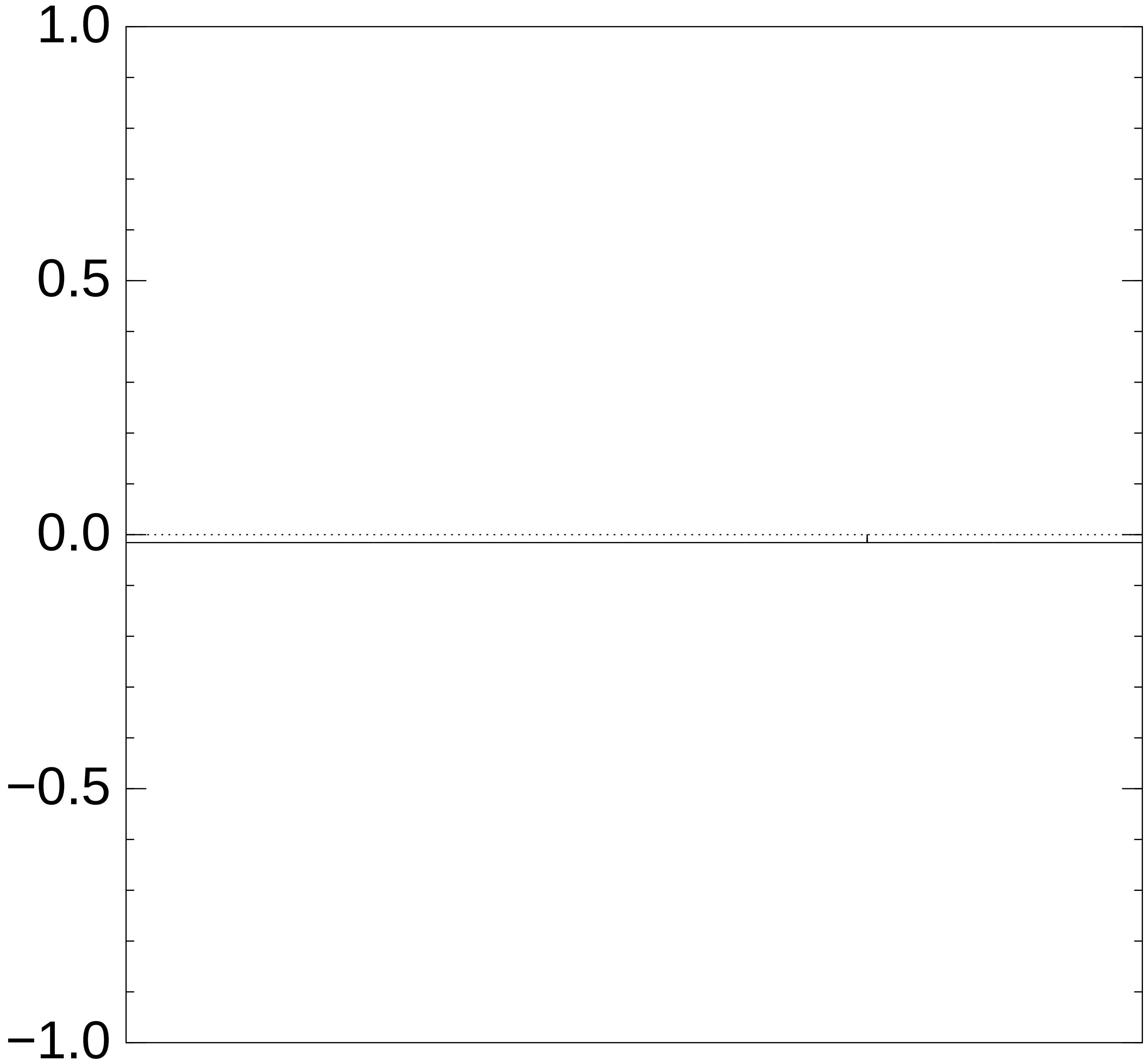


Very bad stopping point.

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

Step $1 +$ Step 2
act linearly on this vector.

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



Very bad stopping point.

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

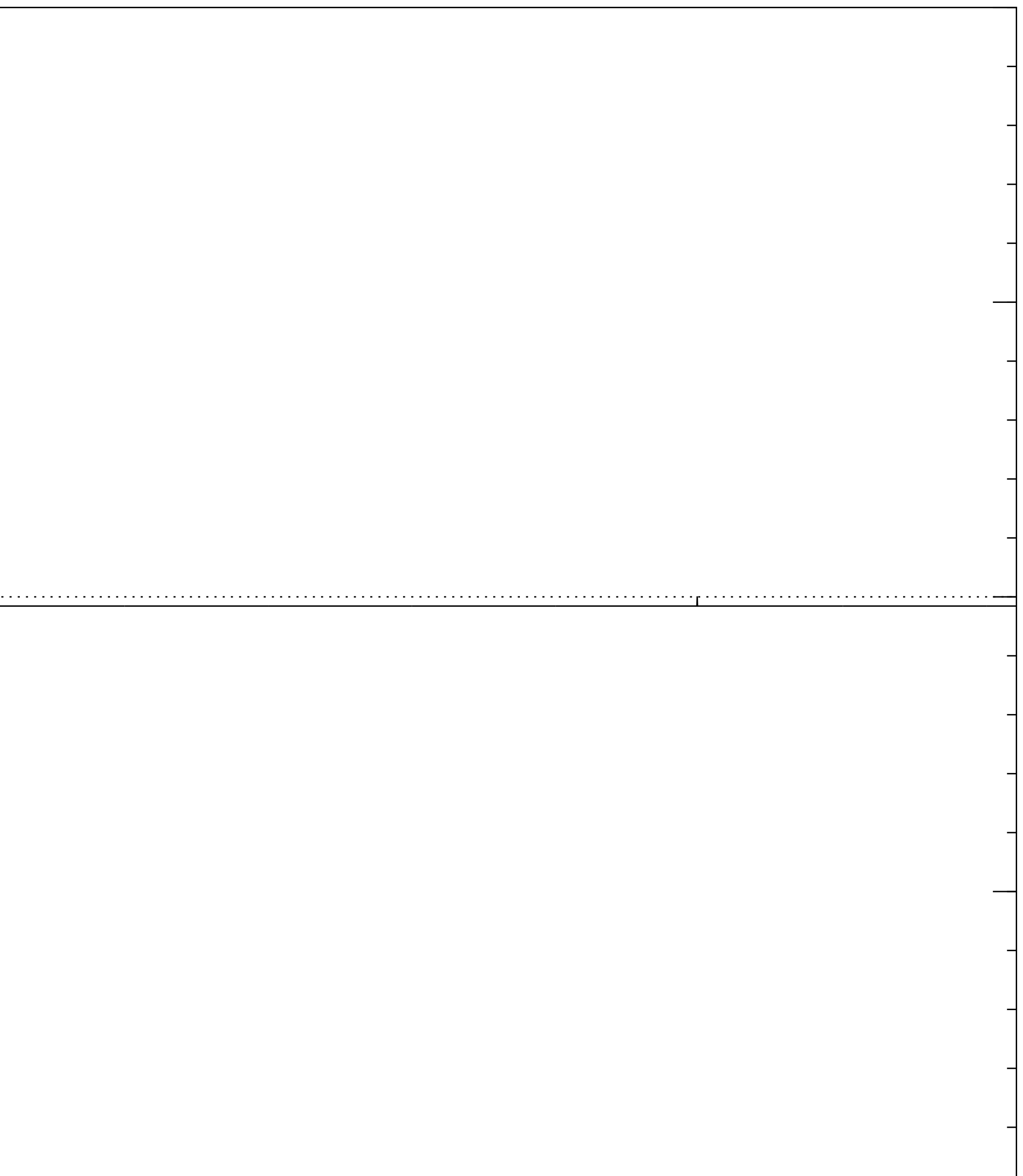
Step 1 + Step 2
act linearly on this vector.

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 $u \mapsto a_u$

xample with $n = 12$

$0 \times$ (Step 1 + Step 2):



d stopping point.

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

(1) a_u for roots u;

(2) a_u for non-roots u.

Step 1 + Step 2 act linearly on this vector.

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

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

Many m

Shor ger
e.g., pol
"cycloto
STOC 2
encrypti

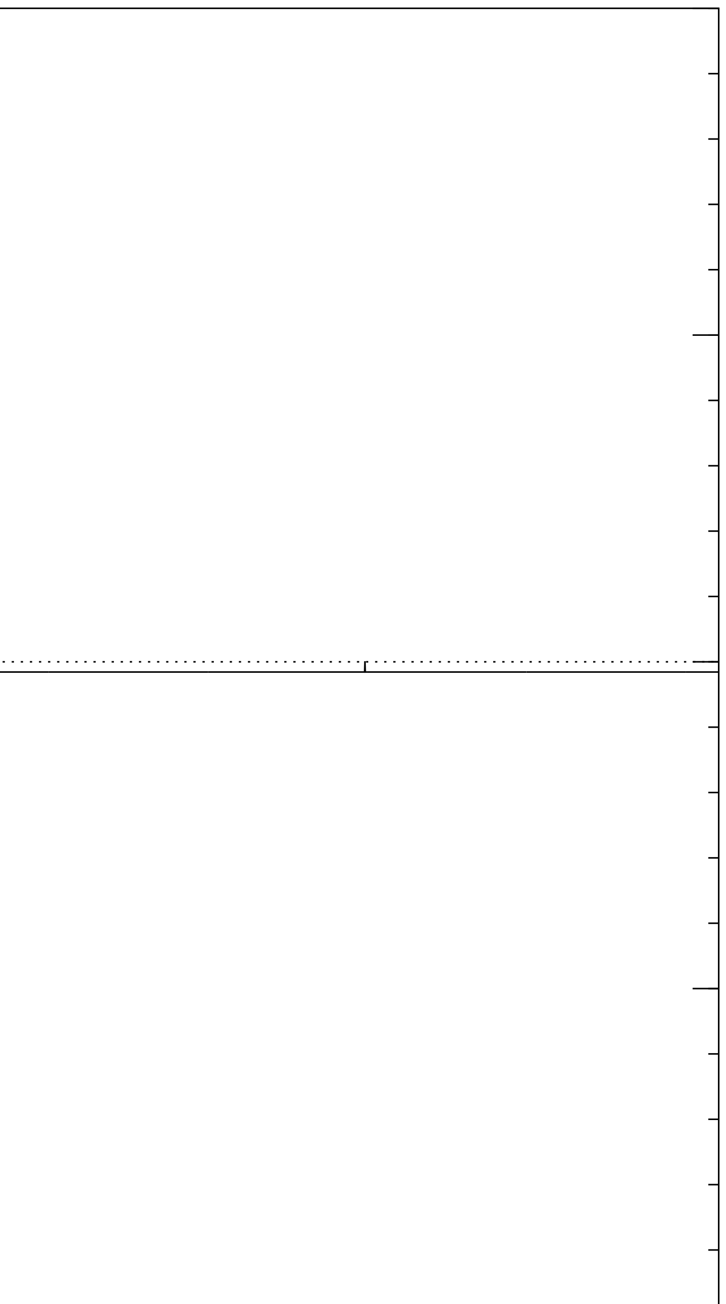
Grover g
e.g., fast
use "qua

Not just
e.g., sub
CRS/CS
uses "Ku

of $u \mapsto a_u$

th $n = 12$

1 + Step 2):



point.

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

Step 1 + Step 2
act linearly on this vector.

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

Many more applic

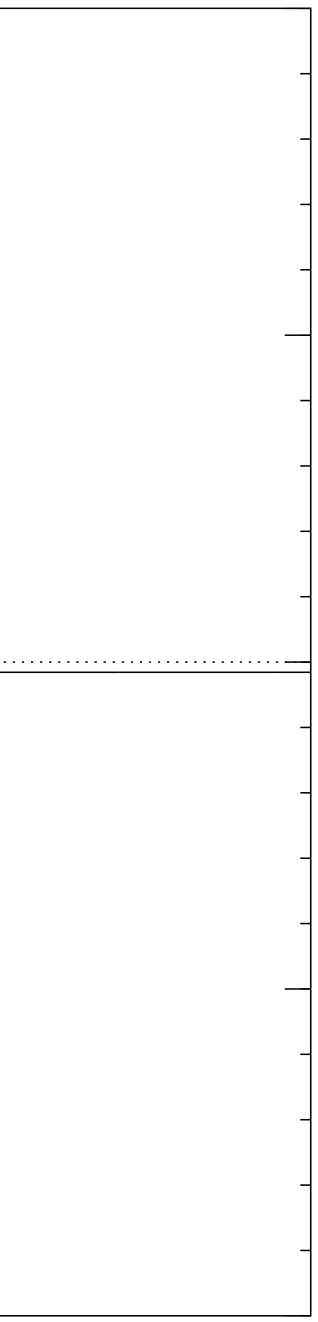
Shor generalizatio
e.g., poly-time att
"cyclotomic" case
STOC 2009 "Fully
encryption using i

Grover generalizat
e.g., fastest subse
use "quantum wal

Not just Shor and
e.g., subexponenti
CRS/CSIDH isoge
uses "Kuperberg's

u

2):



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

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.

Many more applications

Shor generalizations:
e.g., poly-time attack breaki
"cyclotomic" case of Gentry
STOC 2009 "Fully homomo
encryption using ideal lattice

Grover generalizations:
e.g., fastest subset-sum atta
use "quantum walks".

Not just Shor and Grover:
e.g., subexponential-time
CRS/CSIDH isogeny attack
uses "Kuperberg's algorithm

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

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.

Many more applications

Shor generalizations:
e.g., poly-time attack breaking "cyclotomic" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices".

Grover generalizations:
e.g., fastest subset-sum attacks use "quantum walks".

Not just Shor and Grover:
e.g., subexponential-time CRS/CSIDH isogeny attack uses "Kuperberg's algorithm".