Quantum cryptanalysis

Daniel J. Bernstein

Main question in quantum cryptanalysis: What is the most efficient quantum algorithm to attack this cryptosystem?

(For comparison, main question in non-quantum cryptanalysis: What is the most efficient non-quantum algorithm to attack this cryptosystem?)

"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?

(Something to think about: Do we really know the answer for non-quantum computers?)

m cryptanalysis

. Bernstein

estion in
n cryptanalysis:
the most efficient
n algorithm
k this cryptosystem?

nparison, main question
uantum cryptanalysis:
the most efficient
ntum algorithm
k this cryptosystem?)

“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?

(Something to think about:
Do we really know the answer
for non-quantum computers?)

Quantur
contains
can effic
“NOT g
“control

alysis

n

---

lysis:
efficient
n
otosystem?

main question
ryptanalysis:
efficient
rithm
otosystem?)

"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?

(Something to think about:
Do we really know the answer
for non-quantum computers?)

Quantum compute
contains many "qu
can efficiently perf
"NOT gate", "Ha
"controlled NOT g

?

tion

is:

?)

1

"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?

(Something to think about:
Do we really know the answer
for non-quantum computers?)

2

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

“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?

(Something to think about: Do we really know the answer for non-quantum computers?)

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

"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?

(Something to think about:
Do we really know the answer
for non-quantum 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 today.

"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?

(Something to think about:
Do we really know the answer
for non-quantum 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 today.

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

"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?

(Something to think about:
Do we really know the answer
for non-quantum 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 today.

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.

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?

ing to think about:
eally know the answer
quantum 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 today.

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
with as

This is t
quantum
by 1980
paper ap

nm”
m that
ter can run.

instructions,
ction is
puter’s
ion set.

which
antum
pport?

nk about:
the answer
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 today.

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 1980 Manin (E
paper appendix), 1

n.

ns,

er

?)

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 today.

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 1980 Manin (English vers
paper appendix), 1982 Feyn

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 today.

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 1980 Manin (English version:
paper appendix), 1982 Feynman.

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 today.

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 1980 Manin (English version: paper appendix), 1982 Feynman.

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
ain goal of quantum-
er engineering today.

e these instructions
ute "Toffoli gate";
on's algorithm";
r'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 1980 Manin (English version:
paper appendix), 1982 Feynman.

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

Quantu
efficientl
that any
compute

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

structions work
of quantum-
ering today.

tructions
li 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 1980 Manin (English version: paper appendix), 1982 Feynman.

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-
ay.

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 1980 Manin (English version: paper appendix), 1982 Feynman.

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 effic

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 1980 Manin (English version: paper appendix), 1982 Feynman.

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 1980 Manin (English version: paper appendix), 1982 Feynman.

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 1980 Manin (English version:
paper appendix), 1982 Feynman.

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.

4

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

he original concept of
computers introduced
Manin (English version:
ppendix), 1982 Feynman.

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

5

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.

State of

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

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

l concept of
rs introduced
nglish version:
1982 Feynman.

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.

State of a non-qua

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

(QC2):

esired.

of

ced

sion:

man.

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.

State of a non-quantum con

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

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

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

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

State of a non-quantum computer

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

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

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

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

State of a non-quantum computer

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

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.

State of a non-quantum 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.: $(0, 1, 1)$.

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.

State of a non-quantum 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 $\{0, 1\}$.

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.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

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
ne laws of quantum
so a QC2 can efficiently
any physical computer.

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

<u>State of a non-quantum computer</u>

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

<u>State of</u>

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

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.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

State of a quantu

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

(QC3):

g

iently.

n QC3.

ently

uter.

n QC1.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

State of a quantum comput

Data stored in 3 qubits:

a list of 8 numbers, not all z

e.g.: $[3, 1, 4, 1, 5, 9, 2, 6]$.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

State of a quantum computer

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

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

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.: $[-2, 7, -1, 8, 1, -8, -2, 8]$.

State of a non-quantum 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.: $(1,1,1,1,1,0,0,0,1,$
$0,0,0,0,0,0,1,1,0,0,0,$
$0,1,0,0,1,0,0,0,0,0,1,$
$1,0,1,0,0,0,1,0,0,0,1,$
$0,0,1,1,1,0,0,1,0,0,0,$
$1,1,0,1,1,0,0,1,0,0,1)$.

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.: $[0,0,0,0,0,1,0,0]$.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

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.:
$[3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3]$.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

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.

State of a non-quantum 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.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

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.

a non-quantum computer

state") stored in 3 bits:

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

$0, 0)$.

$1, 1)$.

$1, 1)$.

ored in 64 bits:

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

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

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

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

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

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

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

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

of numb

antum computer

ored in 3 bits:

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

bits:

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

$0, 0, 0, 1,$

$0, 0, 0,$

$0, 0, 1,$

$0, 0, 1,$

$0, 0, 0,$

$0, 0, 1)$.

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

mputer

its:

.

}.

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

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.

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
- "collapses" the state.

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
- "collapses" 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 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
- "collapses" the state.

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

"Collapse": New state is all zeros except 1 at position q.

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.:
., 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
- "collapses" the state.

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

"Collapse": New state is all zeros
except 1 at position q.

e.g.: Say
[1, 1, 1, 1

m 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
- "collapses" the state.

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

"Collapse": New state is all zeros
except 1 at position q.

e.g.: Say 3 qubits

$[1, 1, 1, 1, 1, 1, 1, 1]$

er

zero.

, 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
- "collapses" the state.

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

"Collapse": New state is 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
- "collapses" the state.

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

"Collapse": New state is 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
- "collapses" the state.

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

"Collapse": New state is 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
- "collapses" the state.

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

"Collapse": New state is 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
- "collapses" the state.

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

"Collapse": New state is 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

pers stored in n qubits.

ing n qubits

ces n bits and

oses" the state.

its have state

$\ldots, a_{2^n-1}]$ then

ment produces q

bability $|a_q|^2 / \sum_r |a_r|^2$.

se": New state is 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

$[3, 1, 4, 1$

tum computer

t a bit.

k at the list

in n qubits.

its

and

state.

ate

then

luces q

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

state is all zeros

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.

$a_r|^2$.

zeros

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 stat

$[3, 1, 4, 1, 5, 9, 2, 6]$.

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].

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."

: 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

., 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.

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:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[4, 1, 3, 1, 2, 6, 5, 9]$.

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, 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:

., $5, 9, 2, 6] \mapsto$

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:

., $5, 9, 2, 6] \mapsto$

., 2, 6, 5, 9].

ate on 3 qubits:

., $5, 9, 2, 6] \mapsto$

5, 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. C_1N

[3, 1, 4, 1

[3, 1, 1, 4

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.

Controlled-NOT (CNOT) ga

e.g. C_1NOT_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:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 1, 4, 5, 9, 6, 2]$.

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

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:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 9, 5, 6, 2]$.

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. 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 g

Also kno
controlle

e.g. C_2C
$[3, 1, 4, 1$
$[3, 1, 4, 1$

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. $\mathrm{C}_1\mathrm{NOT}_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 1, 4, 5, 9, 6, 2]$.

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 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. $\mathrm{C}_0\mathrm{C}_1\mathrm{NOT}_2$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 6, 5, 9, 2, 1]$.

ed-NOT (CNOT) gates

IOT$_0$:

$., 5, 9, 2, 6] \mapsto$

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

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$_0$:

$., 5, 9, 2, 6] \mapsto$

$., 9, 5, 6, 2]$.

IOT$_2$:

$., 5, 9, 2, 6] \mapsto$

$5, 5, 1, 2, 1]$.

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, CN
to build other per

ates

nt:
i.e.,
q_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 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. $C_2C_1NOT_0$:
$[3, 1, 4, 1, 5, 9, 2, 6] \mapsto$
$[3, 1, 4, 1, 5, 9, 6, 2]$.

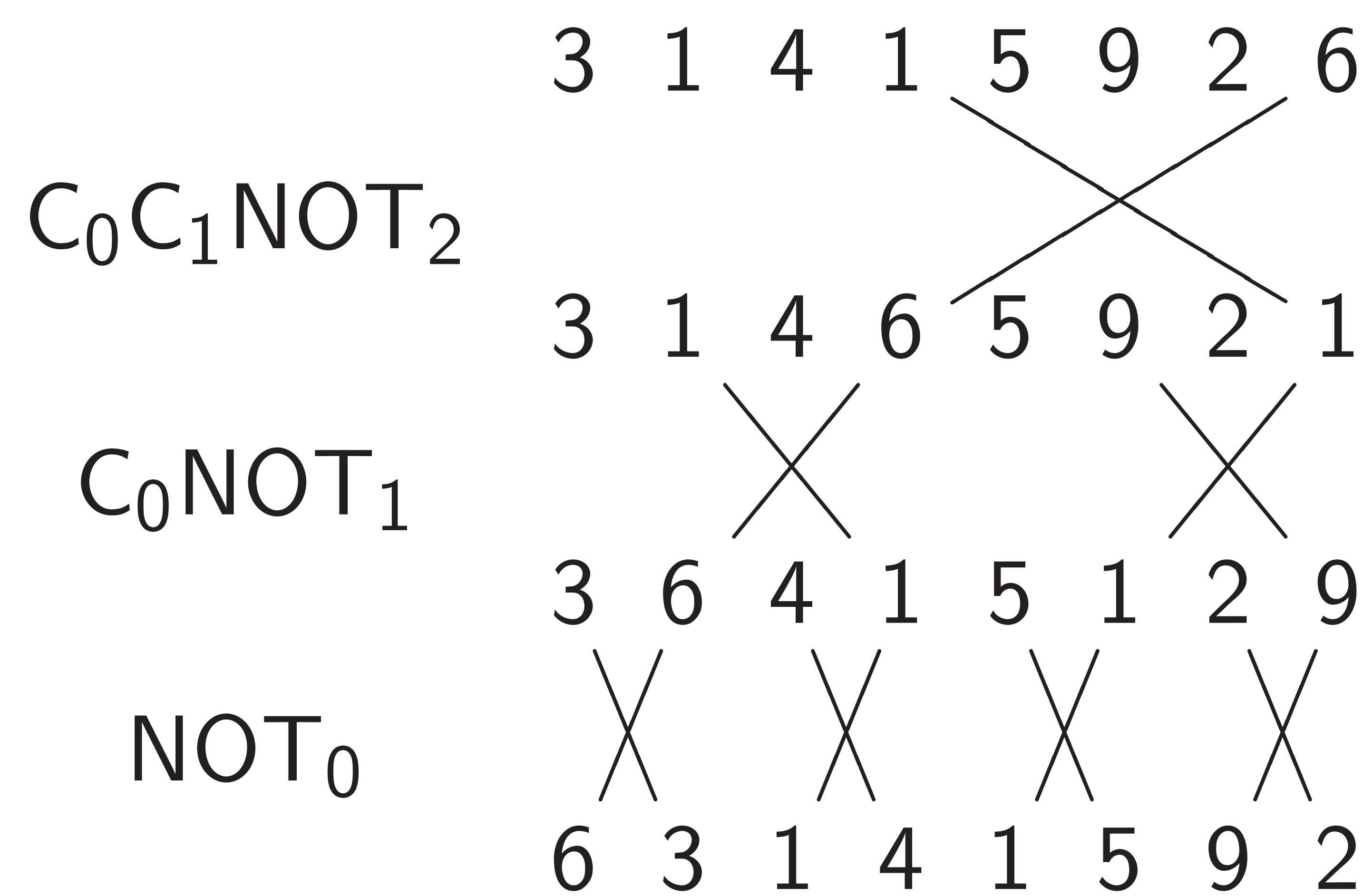
Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

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:



ates

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

C_1NOT_0:
$., 5, 9, 2, 6] \mapsto$
$., 5, 9, 6, 2]$.

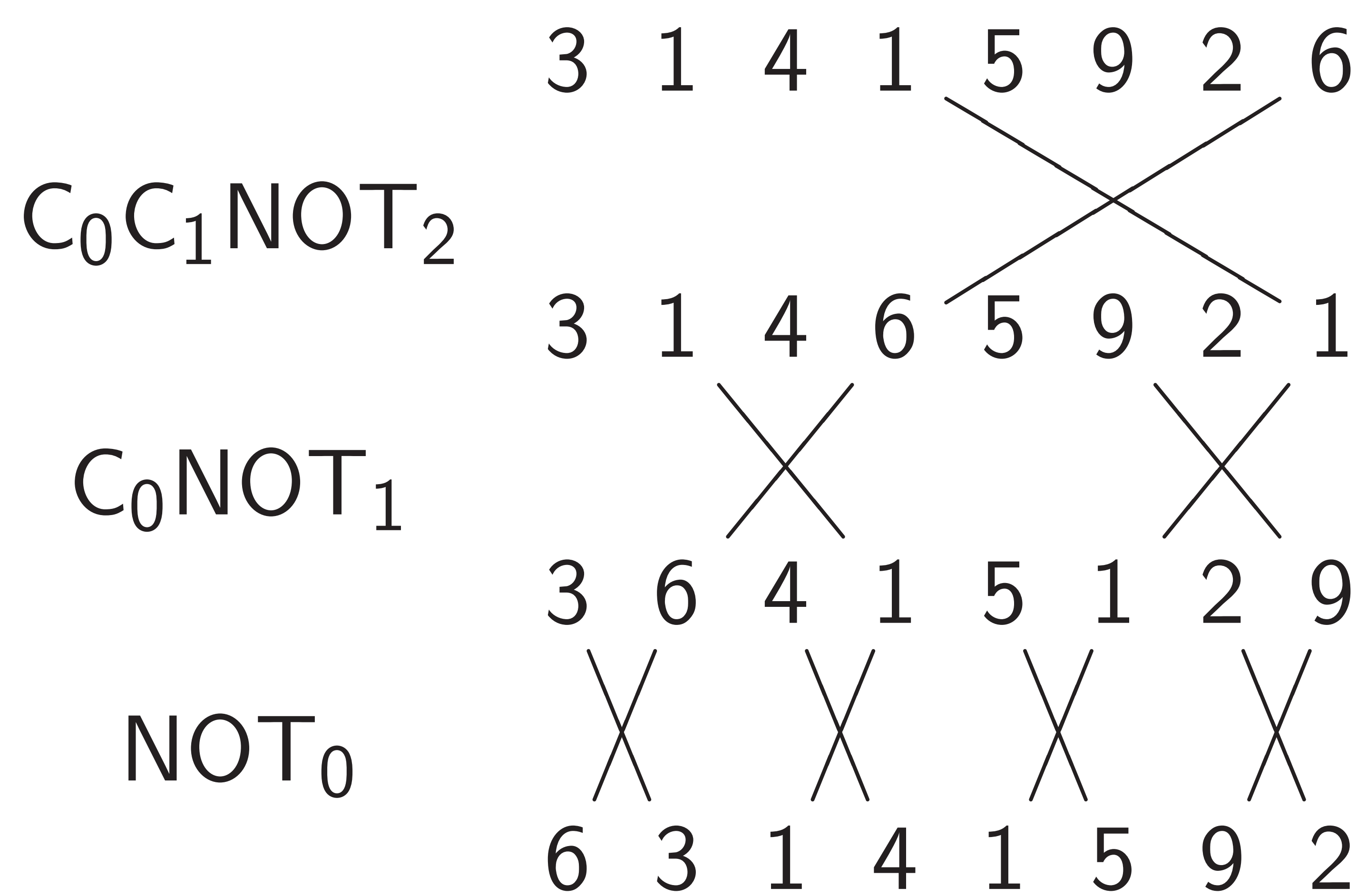
on after measurement:
$q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

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

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

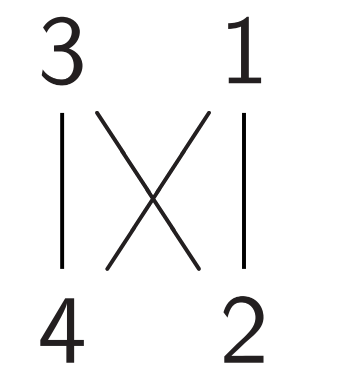
e.g. series of gates to
rotate 8 positions by distance 1:



Hadama

Hadama

$[a, b] \mapsto$



NOT gates:
ed-NOT gates.

| $\mapsto$
|.

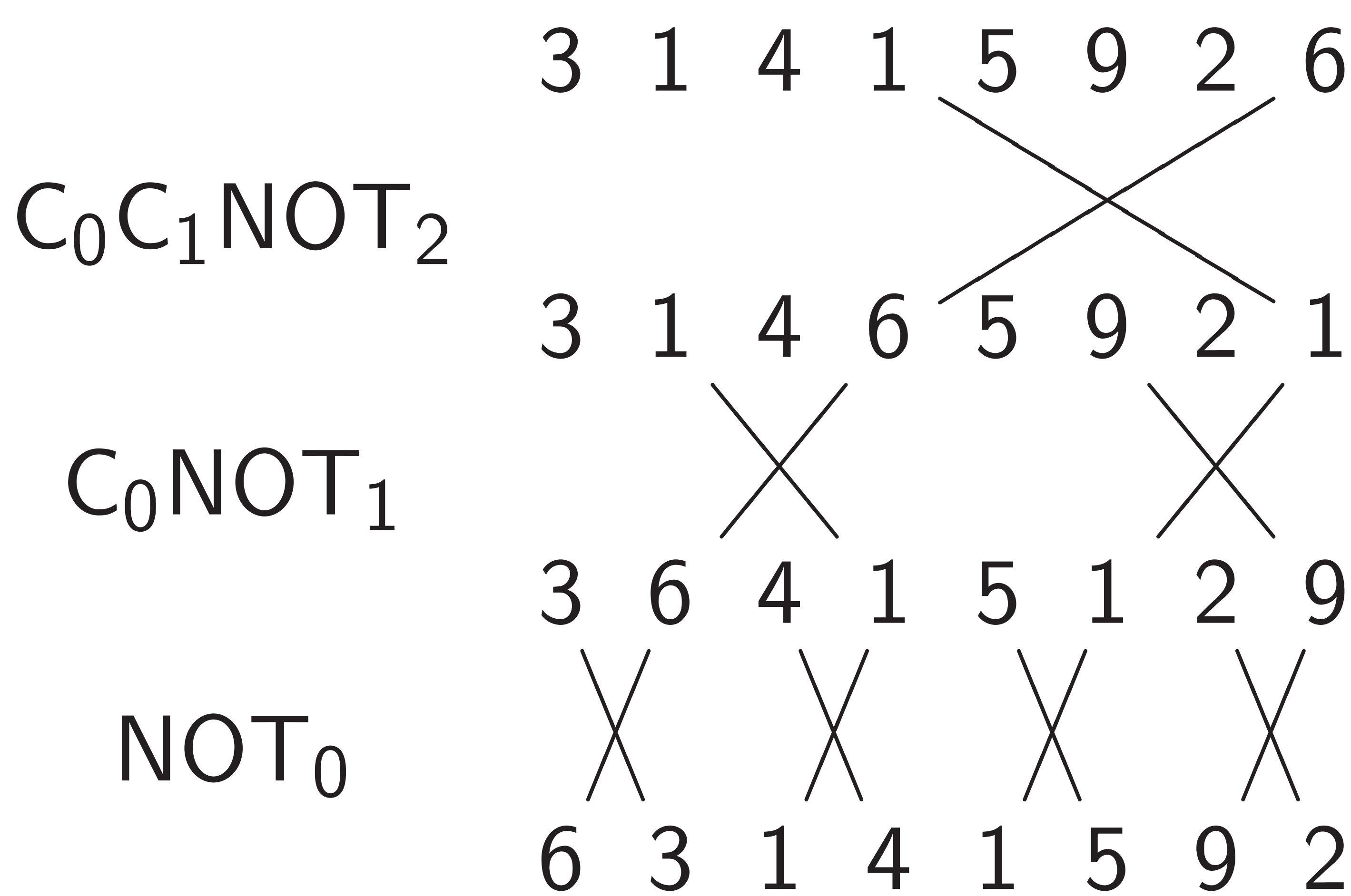
easurement:
$, q_1, q_0 \oplus q_1 q_2)$.

| $\mapsto$
|.

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

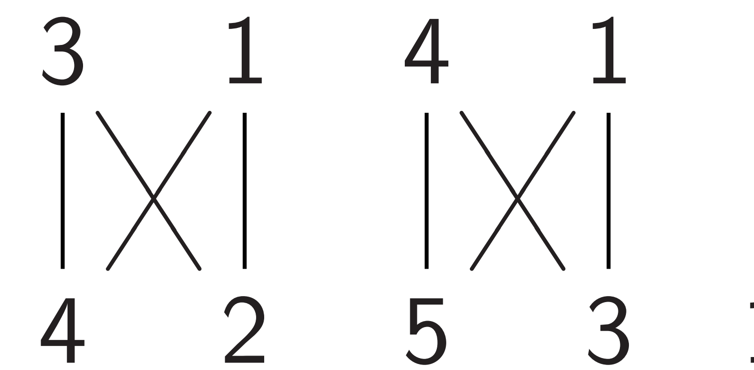
e.g. series of gates to
rotate 8 positions by distance 1:



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:

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:

$[a, b] \mapsto [a + b, a - b]$.

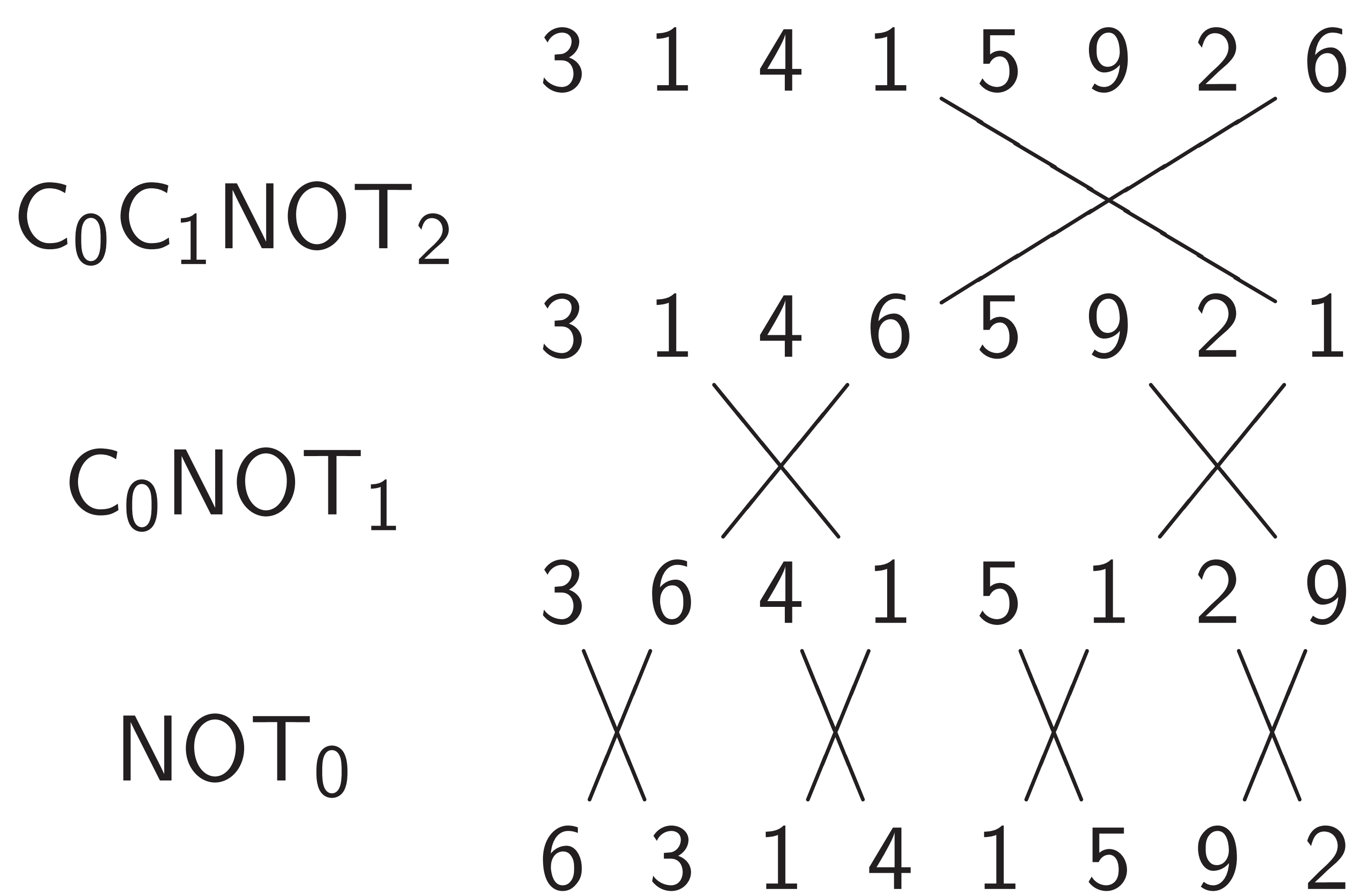
3 1   4 1   5 9   2

4 2   5 3   14 −4   8

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

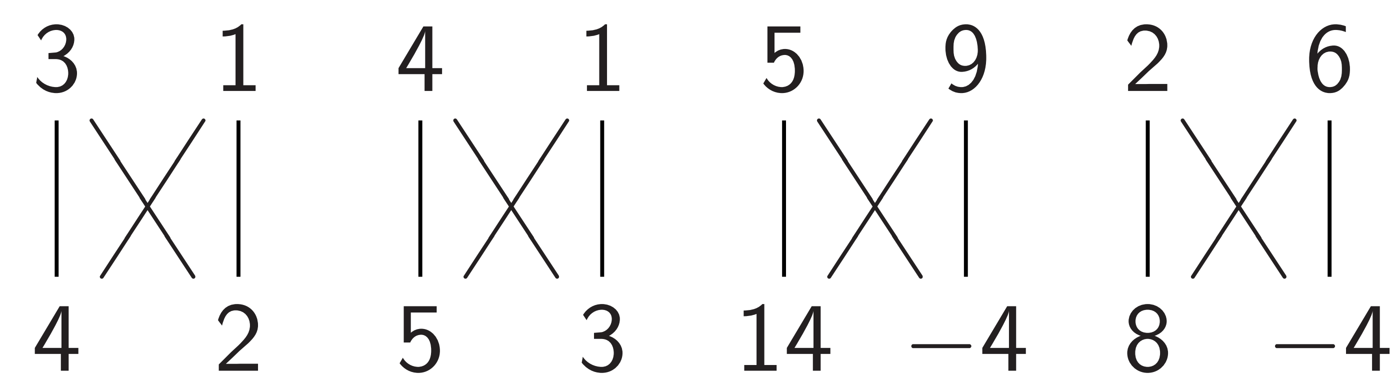
e.g. series of gates to
rotate 8 positions by distance 1:



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

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

$\mathrm{Hadamard}_0$:

$[a, b] \mapsto [a + b, a - b]$.

3 1   4 1   5 9   2 6

4 2   5 3   14 −4   8 −4

$\mathrm{Hadamard}_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.

3 1 4 1   5 9 2 6

7 2 −1 0   7 15 3 3

Some us

Hadama

3 1

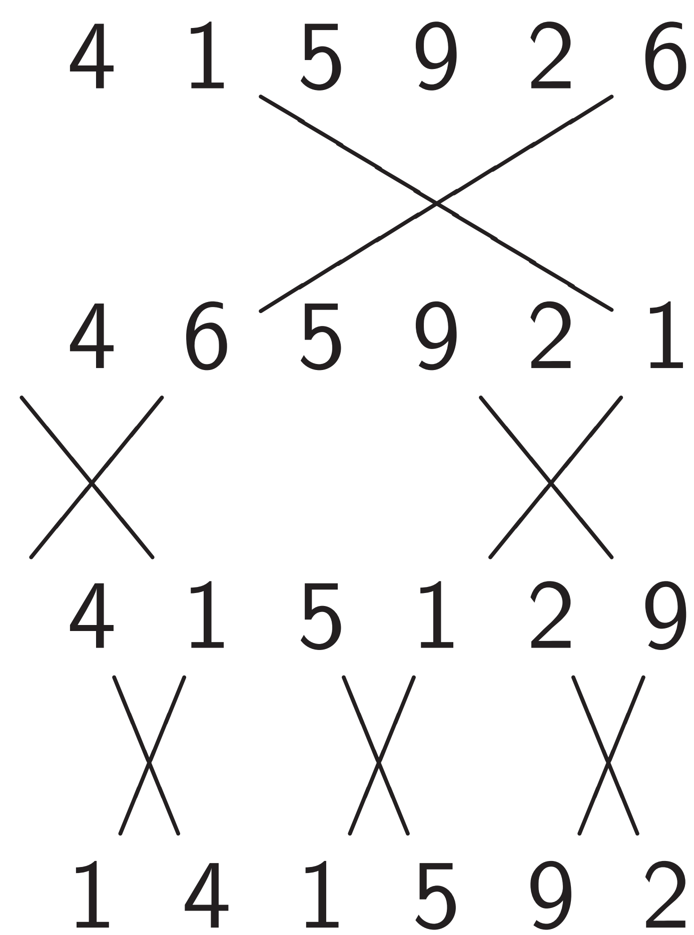
4 2

2 4

6 −2

NOT, Toffoli
mutations.

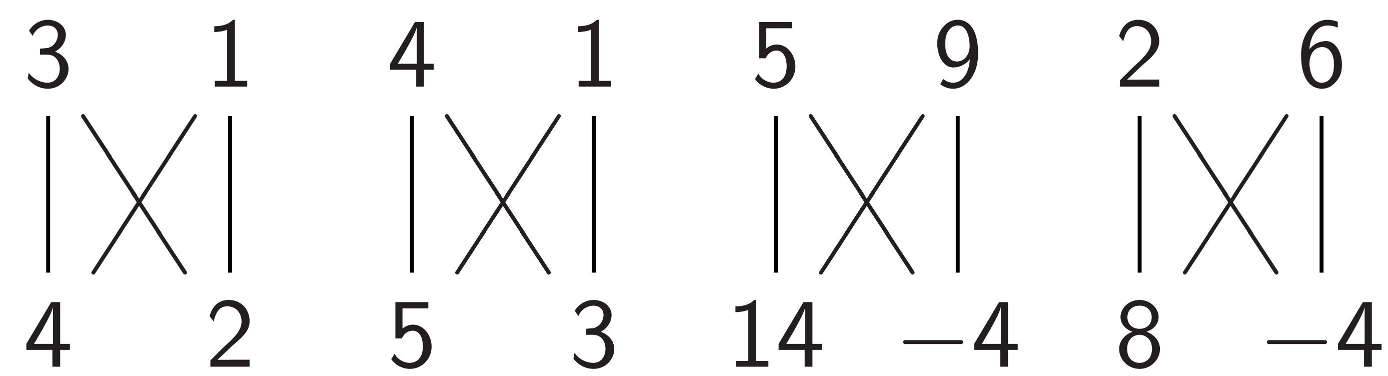
s to
by distance 1:



Hadamard gates

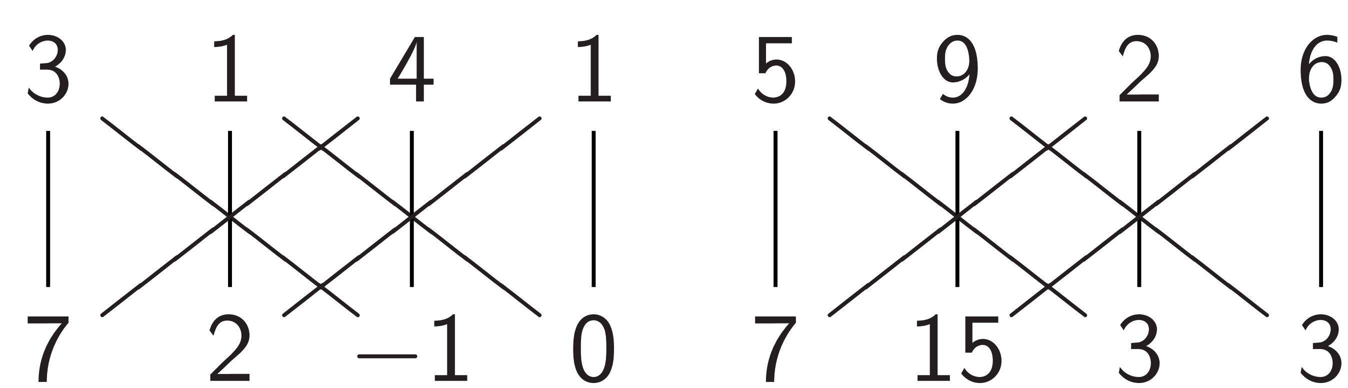
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



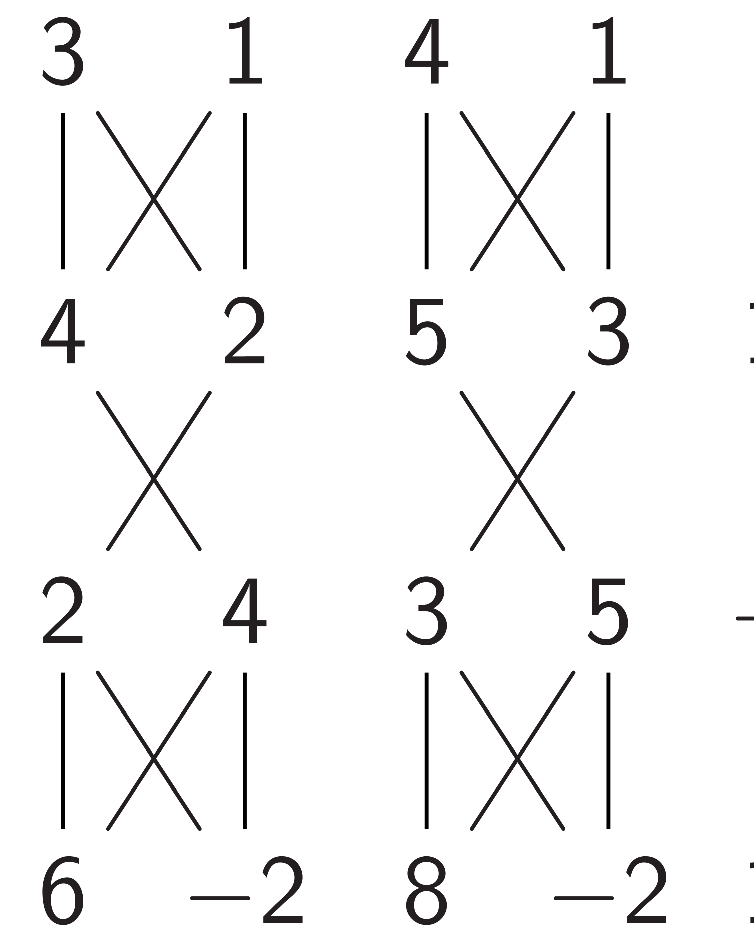
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



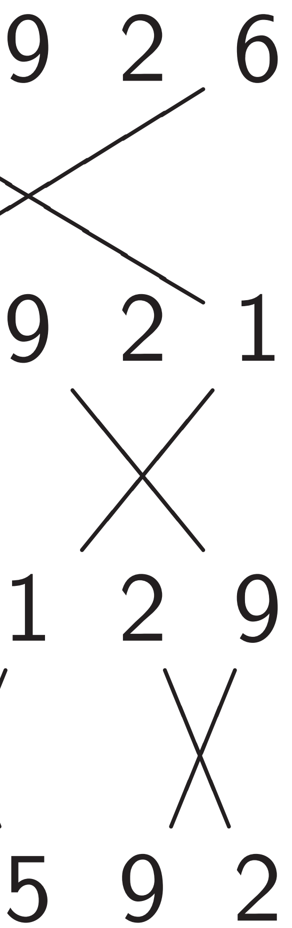
Some uses of Had

Hadamard$_0$, NOT$_($



foli

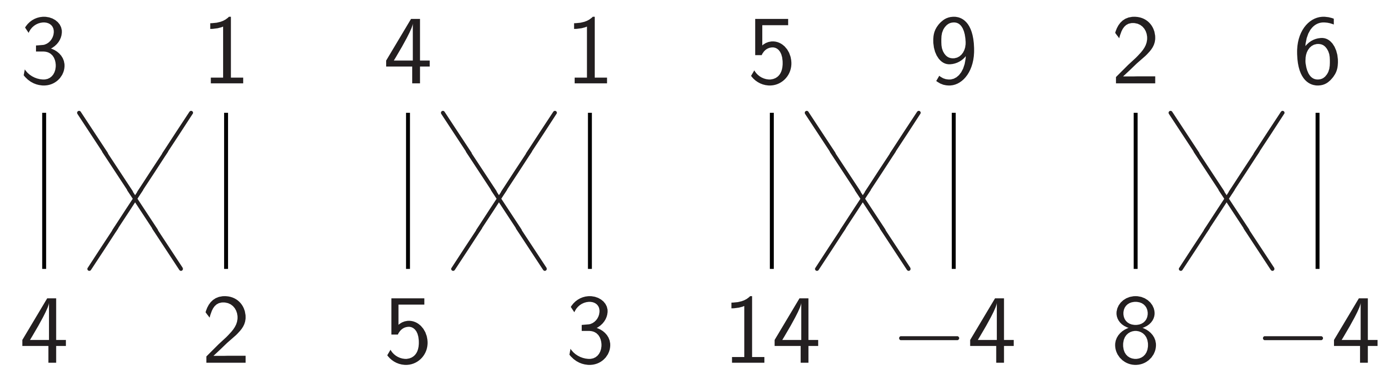
ce 1:



Hadamard gates

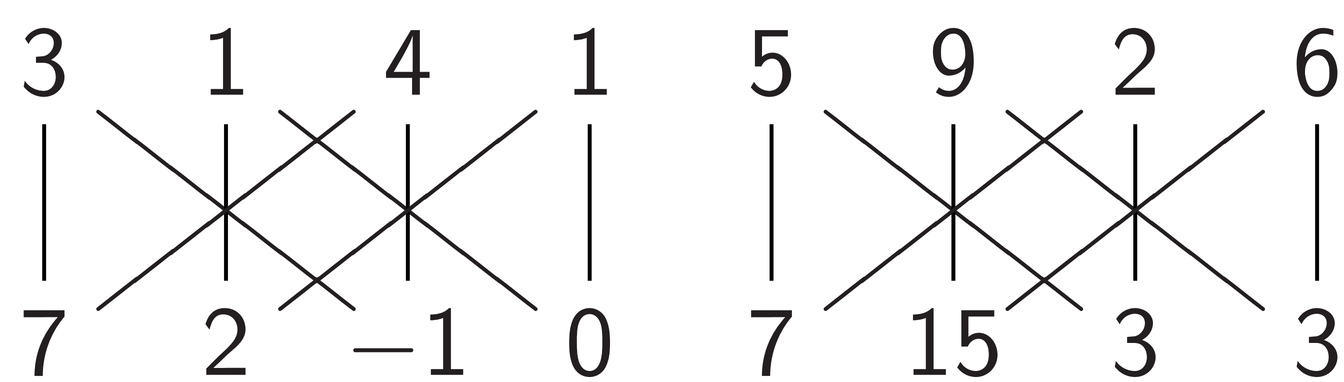
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



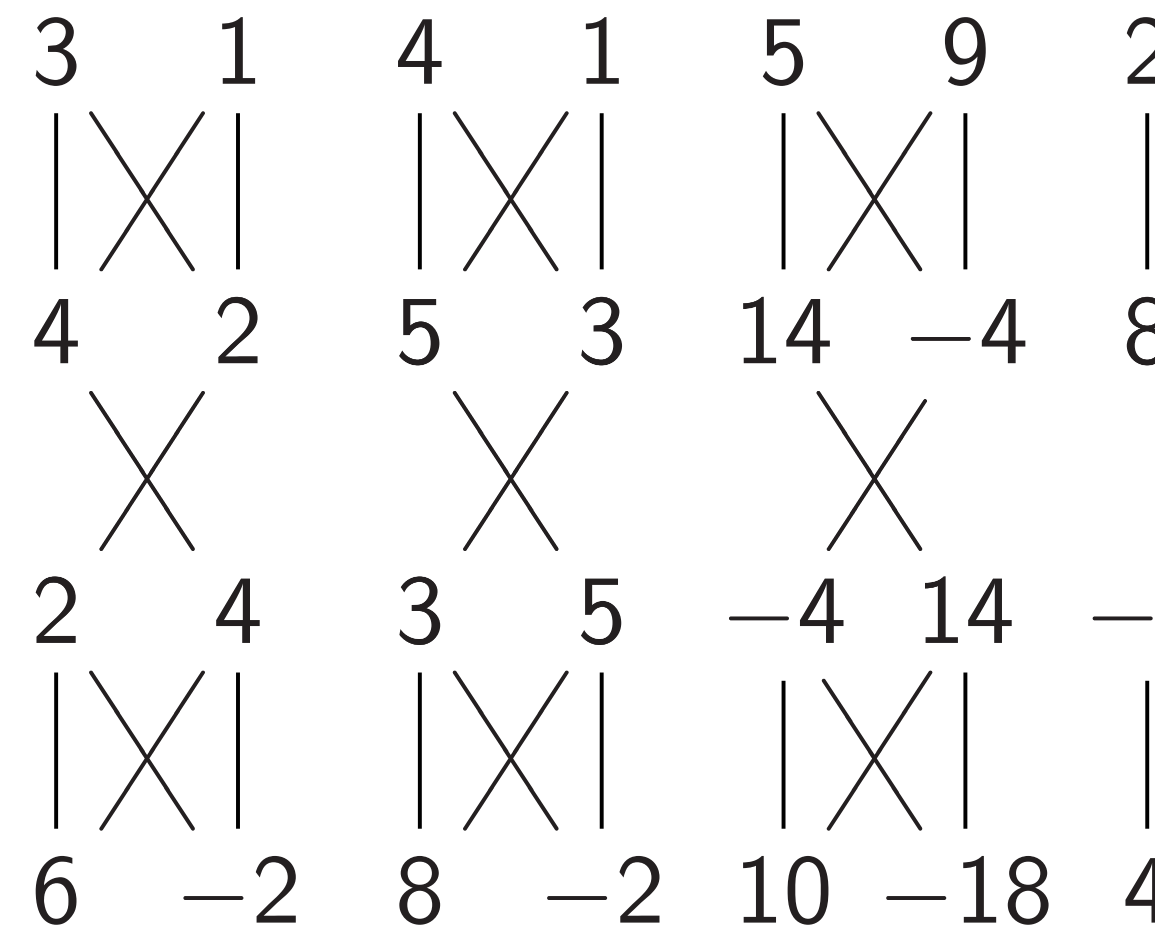
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



Some uses of Hadamard gat

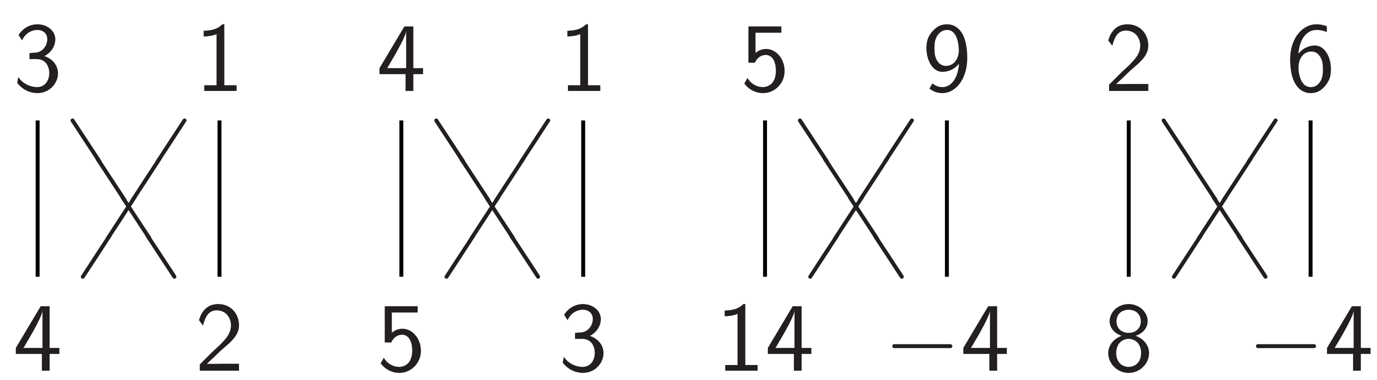
Hadamard$_0$, NOT$_0$, Hadama



Hadamard gates

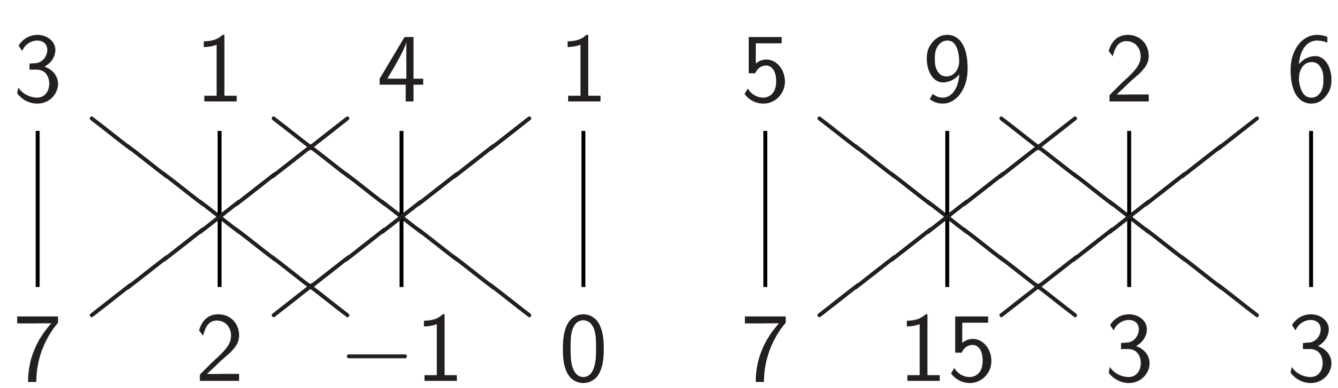
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



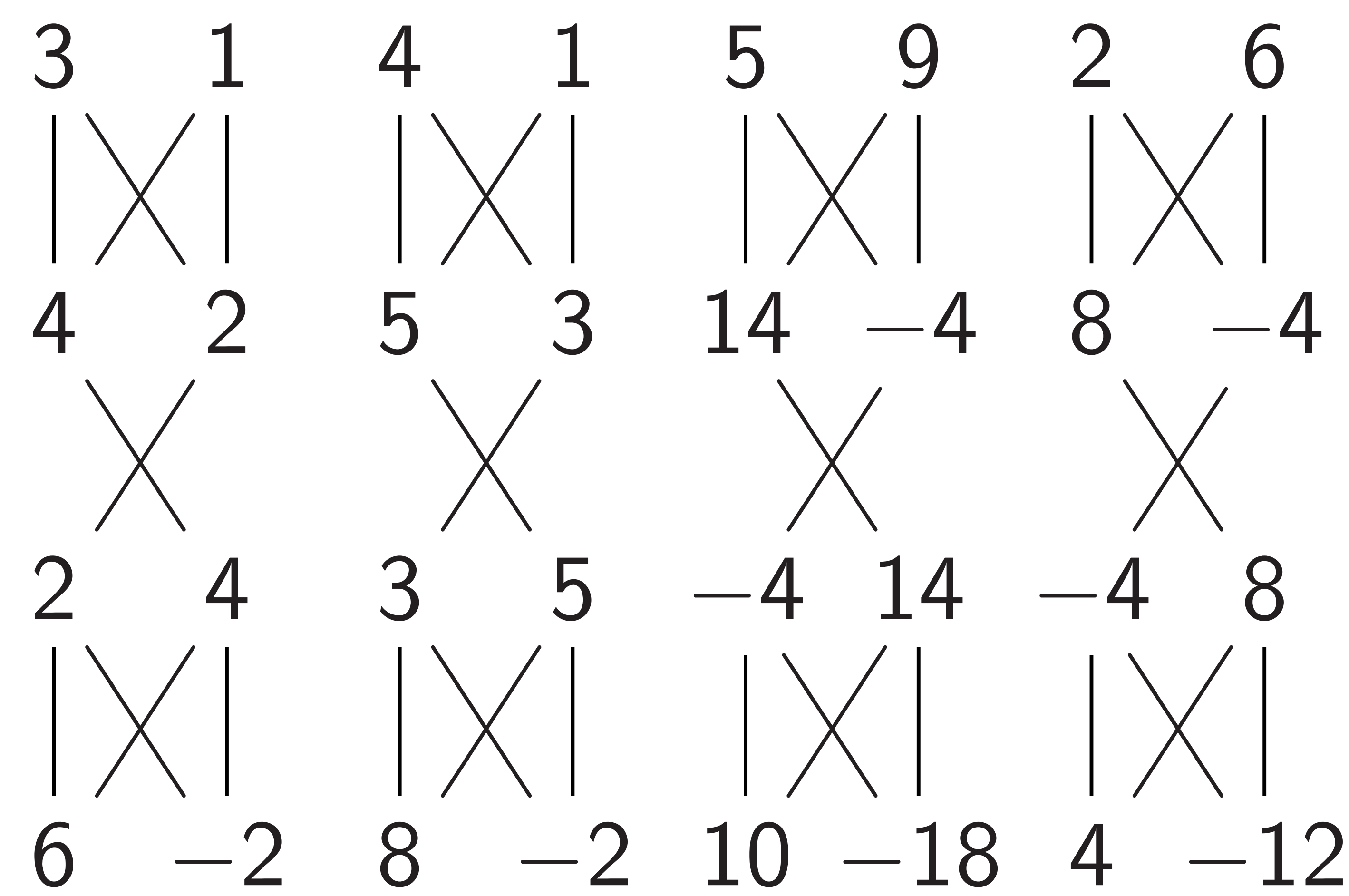
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



Some uses of Hadamard gates

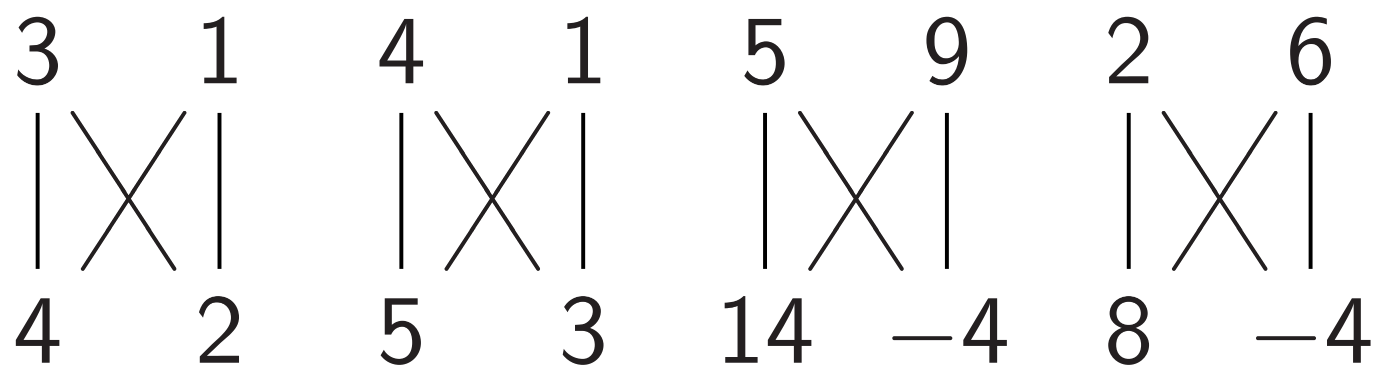
Hadamard$_0$, NOT$_0$, Hadamard$_0$:



Hadamard gates

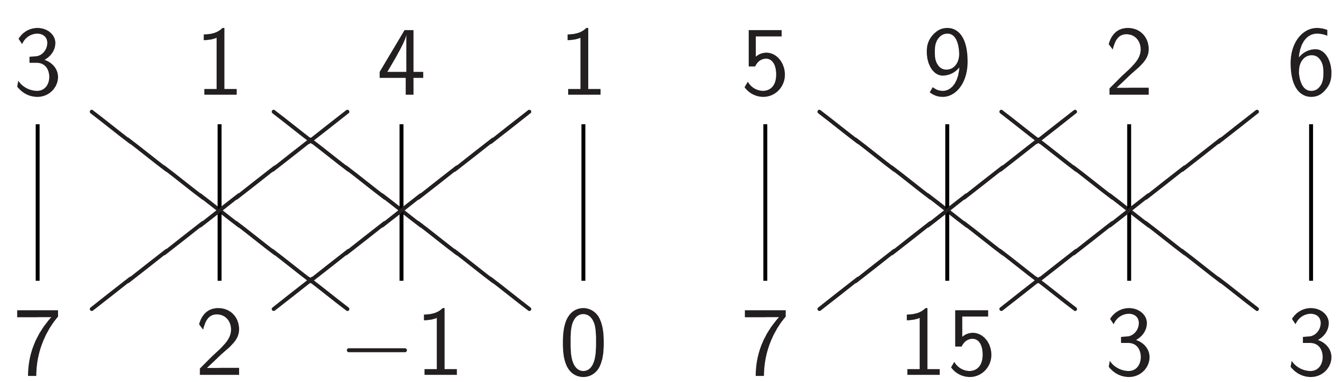
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



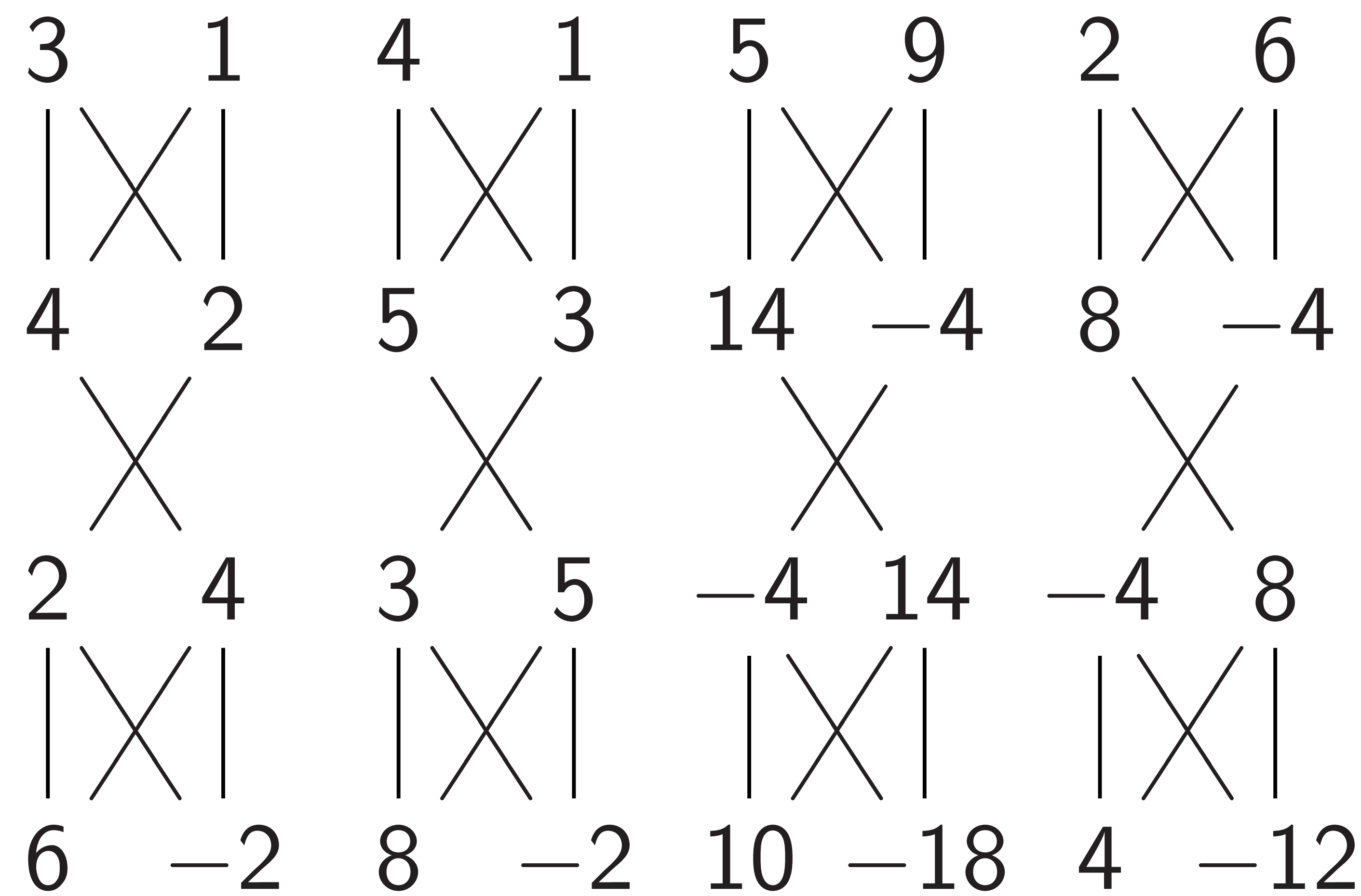
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



Some uses of Hadamard gates

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



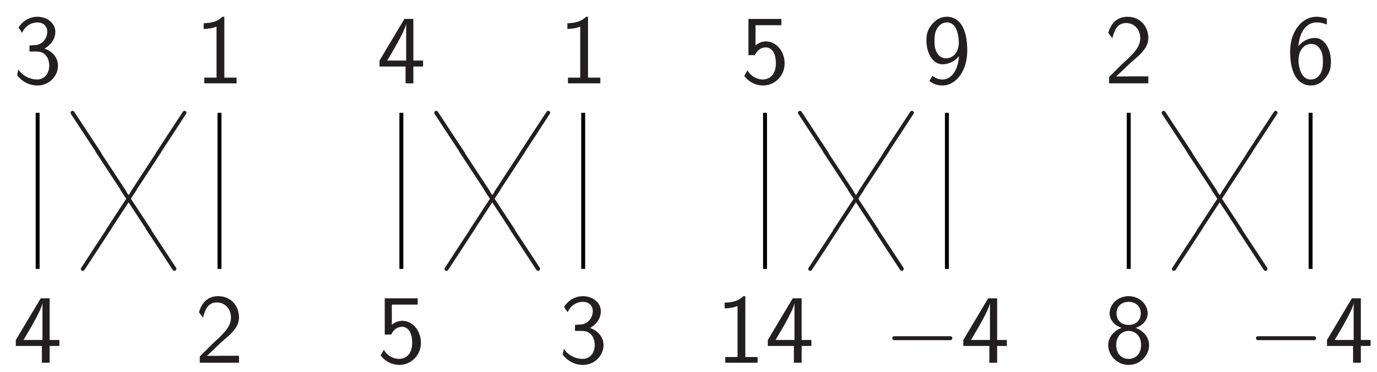
“Multiplied each amplitude by 2.”

This is not physically observable.

Hadamard gates

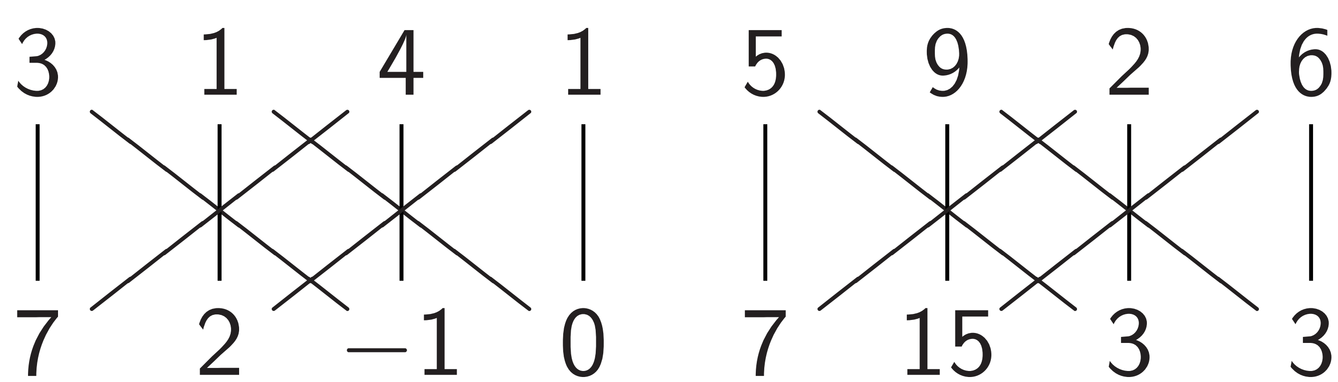
Hadamard$_0$:

$[a, b] \mapsto [a + b, a - b]$.



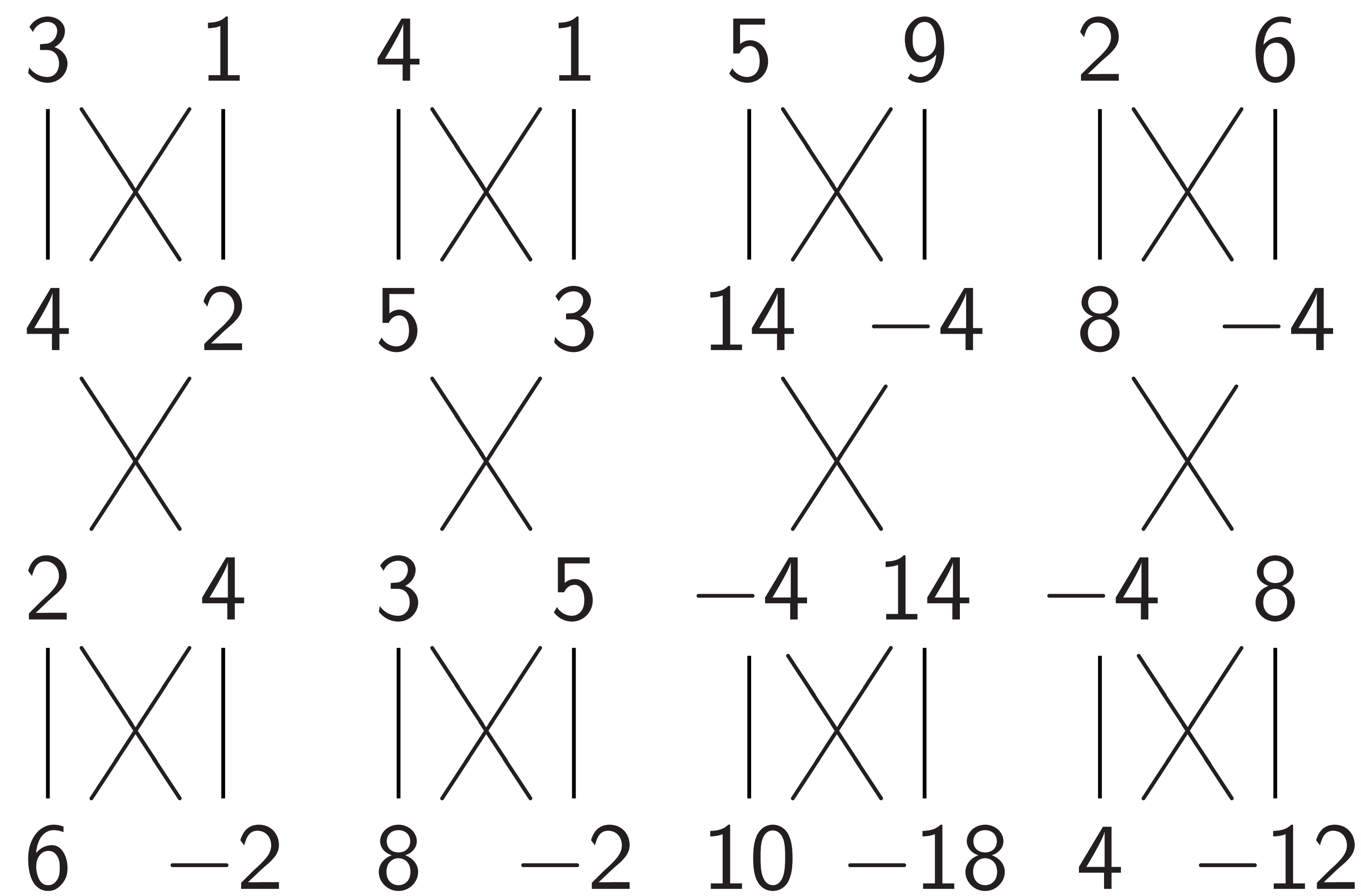
Hadamard$_1$:

$[a, b, c, d] \mapsto$
$[a + c, b + d, a - c, b - d]$.



Some uses of Hadamard gates

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



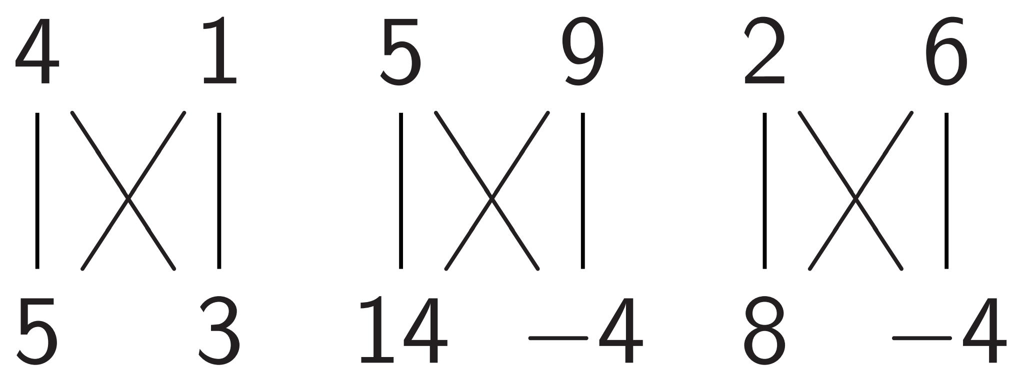
"Multiplied each amplitude by 2."
This is not physically observable.

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

rd gates

rd$_0$:

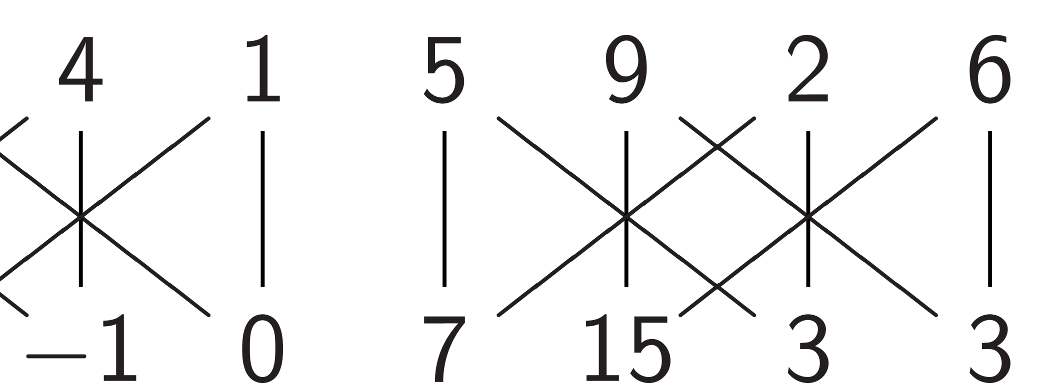
$[a + b, a - b]$.



rd$_1$:

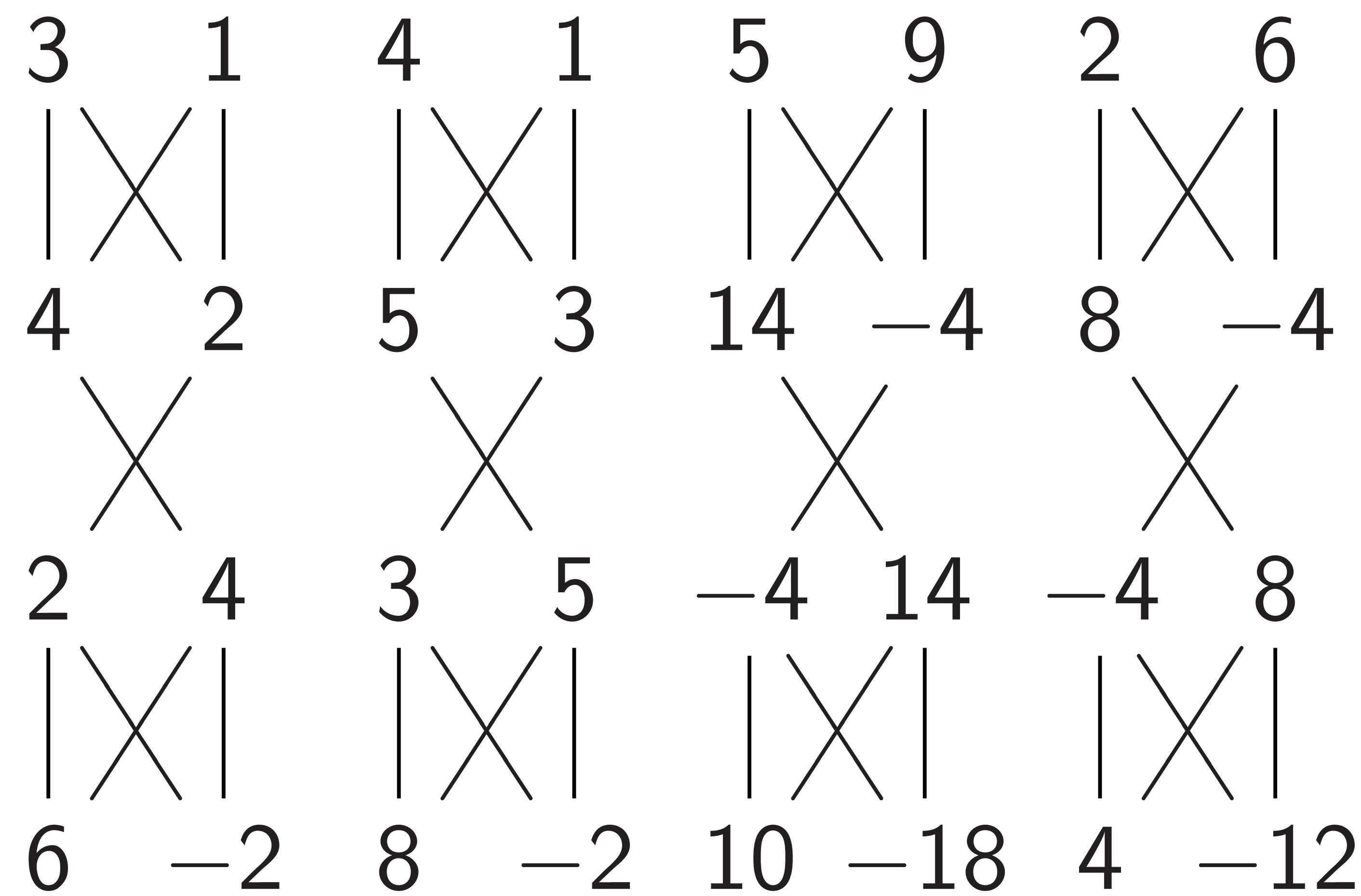
$d] \mapsto$

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



Some uses of Hadamard gates

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



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

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

Fancier

“Negate

Assumes

C_0C_1N

Hadama

NOT

Hadama

C_0C_1N

$- b]$.

5 9 2 6

14 −4 8 −4

$c, b - d]$.

5 9 2 6

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

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

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

Fancier example:

"Negate amplitud

Assumes $q_2 = 0$:

3 1

$C_0C_1NOT_2$

3 1

Hadamard$_2$

3 1

NOT$_2$

3 1

Hadamard$_2$

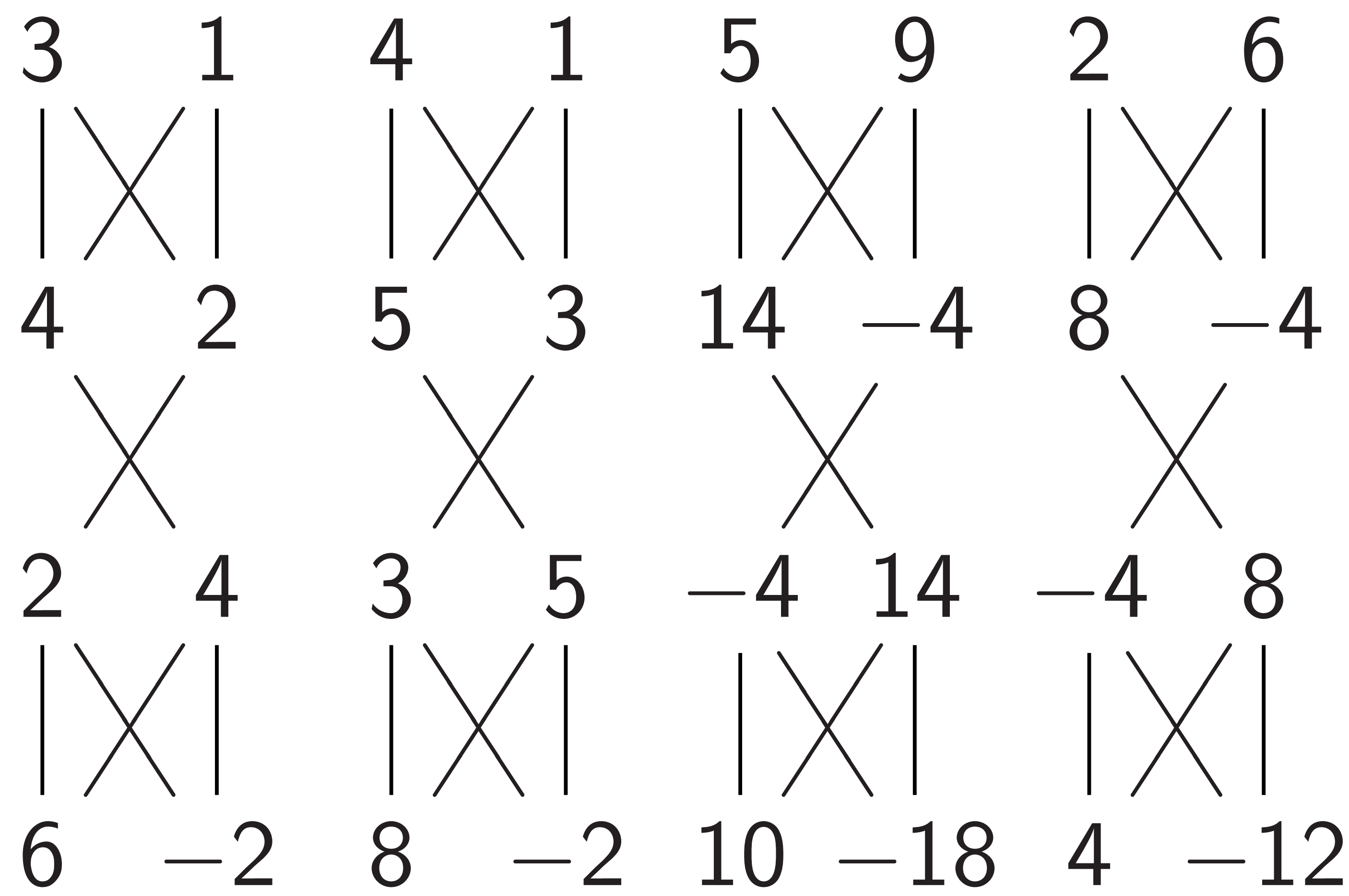
6 2

$C_0C_1NOT_2$

6 2

Some uses of Hadamard gates

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



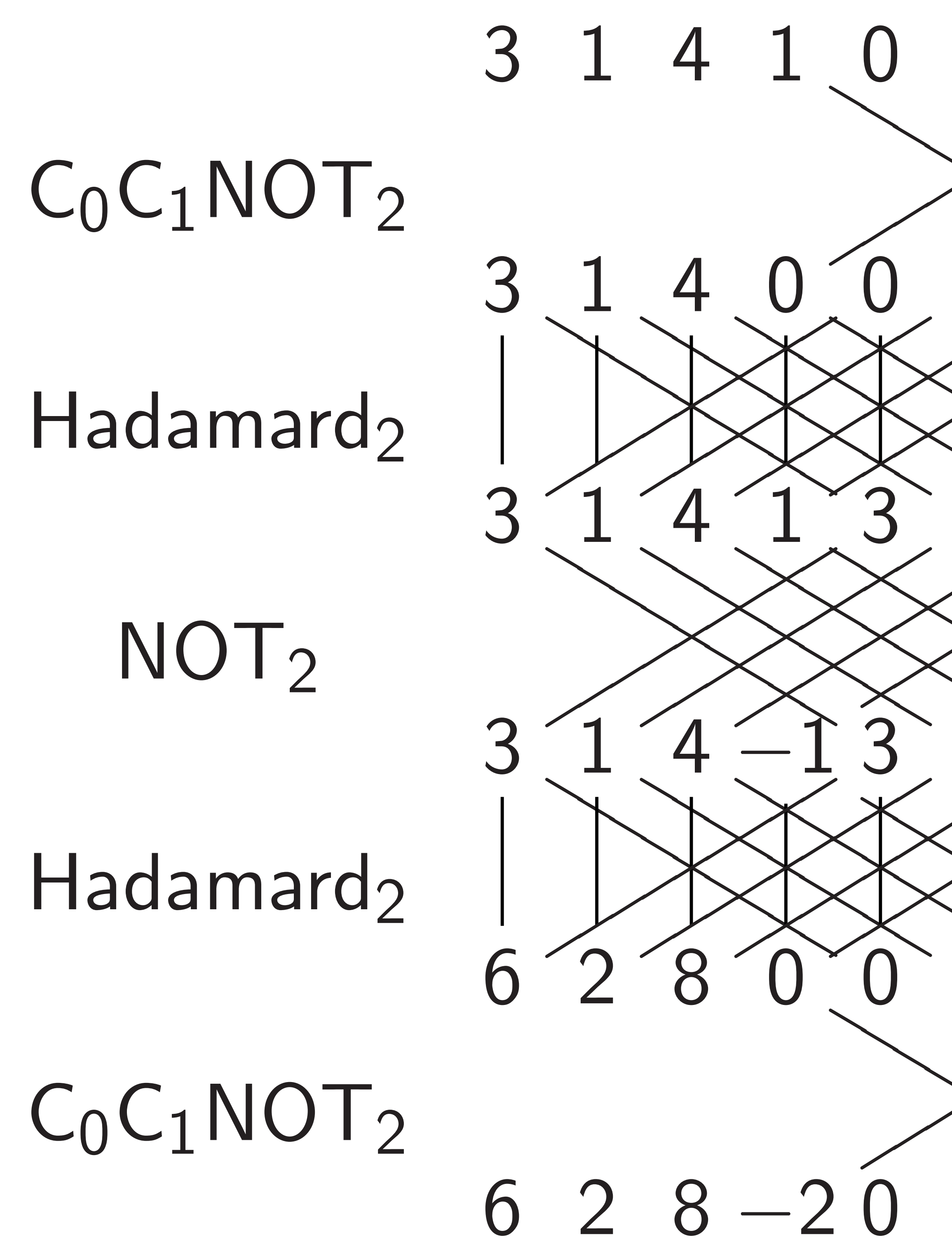
"Multiplied each amplitude by 2."
This is not physically observable.

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

Fancier example:

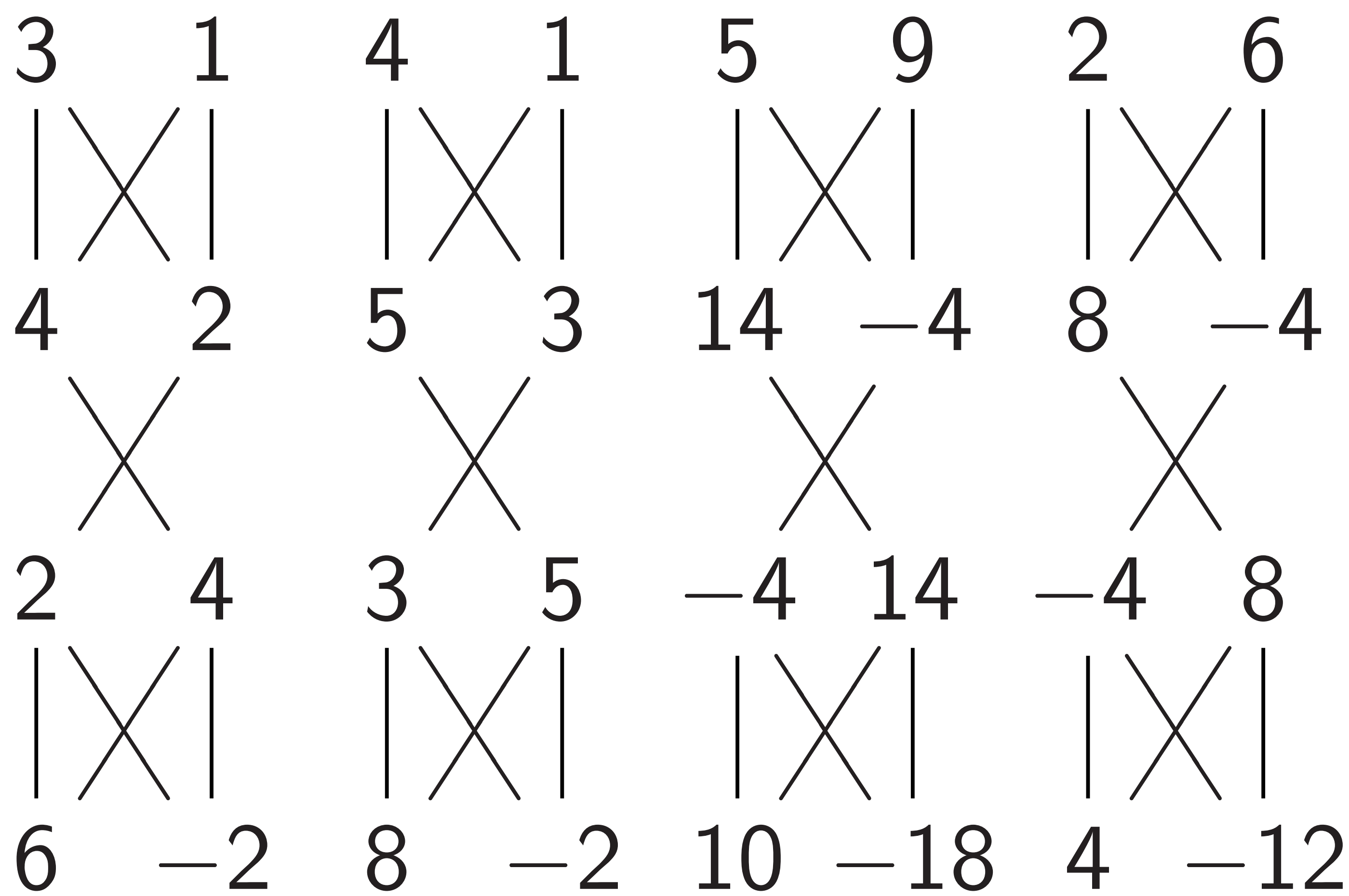
"Negate amplitude if $q_0 q_1$ is

Assumes $q_2 = 0$: "ancilla" q



Some uses of Hadamard gates

Hadamard_0, NOT_0, Hadamard_0:

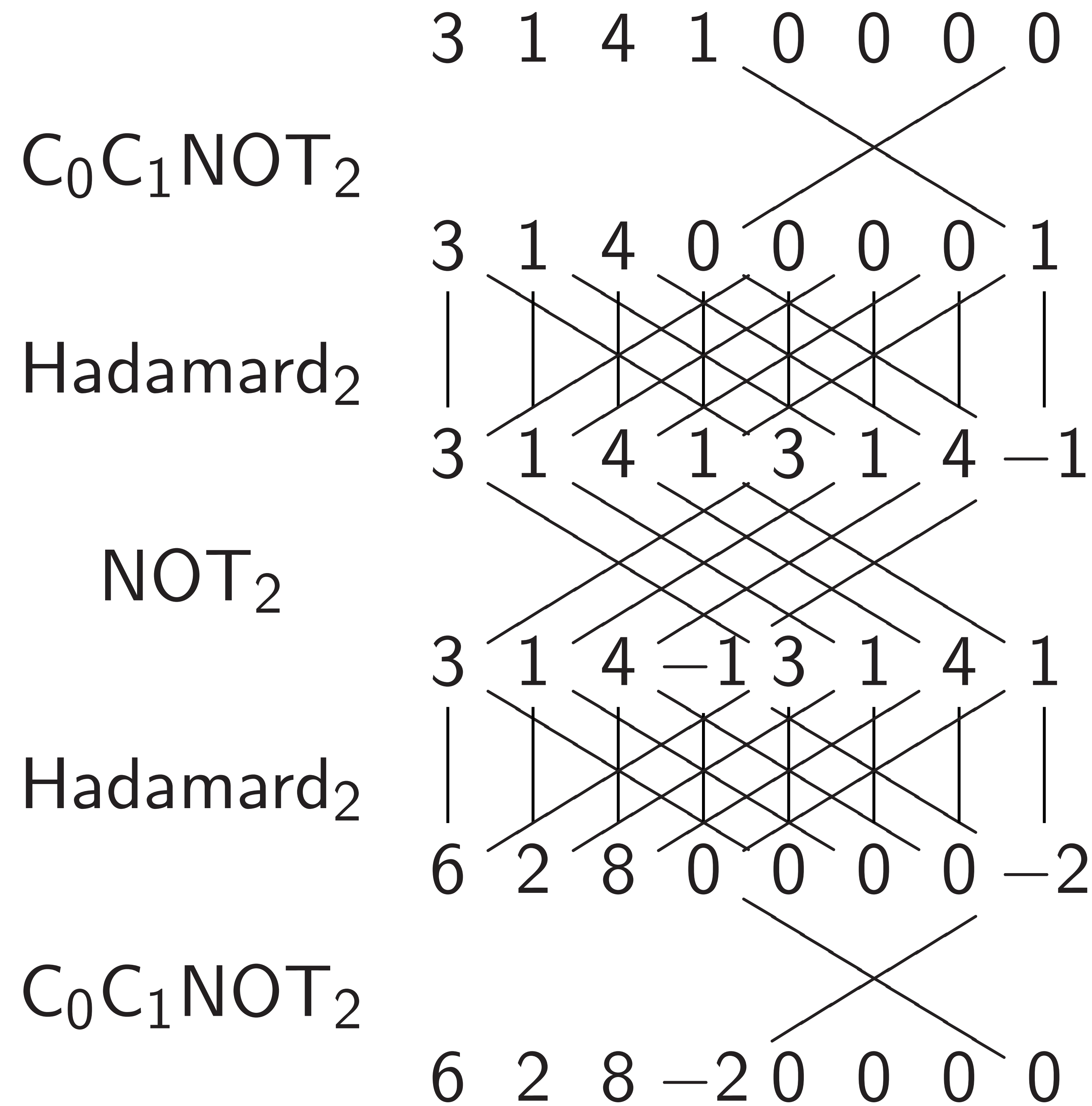


"Multiplied each amplitude by 2."
This is not physically observable.

"Negated 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.



ses of Hadamard gates

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

4	1	5	9	2	6
5	3	14	−4	8	−4
3	5	−4	14	−4	8
8	−2	10	−18	4	−12

ied each amplitude by 2.”
not physically observable.

d 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.

	3	1	4	1	0	0	0	0
C$_0$C$_1$NOT$_2$								
	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$_0$C$_1$NOT$_2$								
	6	2	8	−2	0	0	0	0

Affects r
amplitud
[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

mplitude by 2."

ally observable.

de 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

$\mathrm{C}_0\mathrm{C}_1\mathrm{NOT}_2$

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

$\mathrm{C}_0\mathrm{C}_1\mathrm{NOT}_2$

6 2 8 -2 0 0 0 0

Affects measureme

amplitude around

$[3, 1, 4, 1] \mapsto [1.5,$

es

ard$_0$:

2 6

3 -4

4 8

4 -12

by 2.”

able.

set.”

.

Fancier example:

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

Assumes $q_2 = 0$: “ancilla” qubit.

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

Affects measurements: “Neg

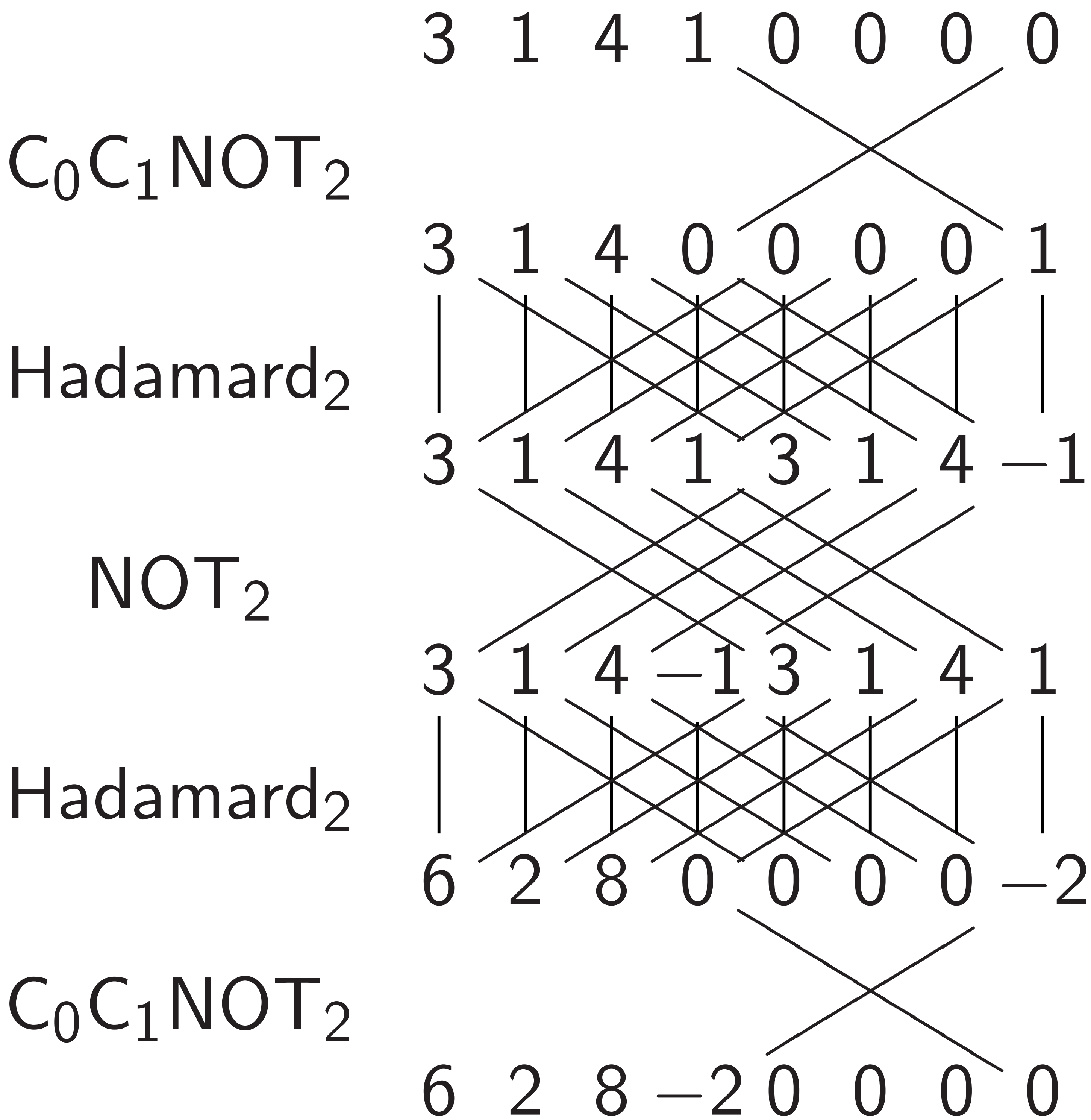
amplitude around its averag

$[3, 1, 4, 1] \mapsto [1.5, 3.5, 0.5, 3.$

Fancier example:

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

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



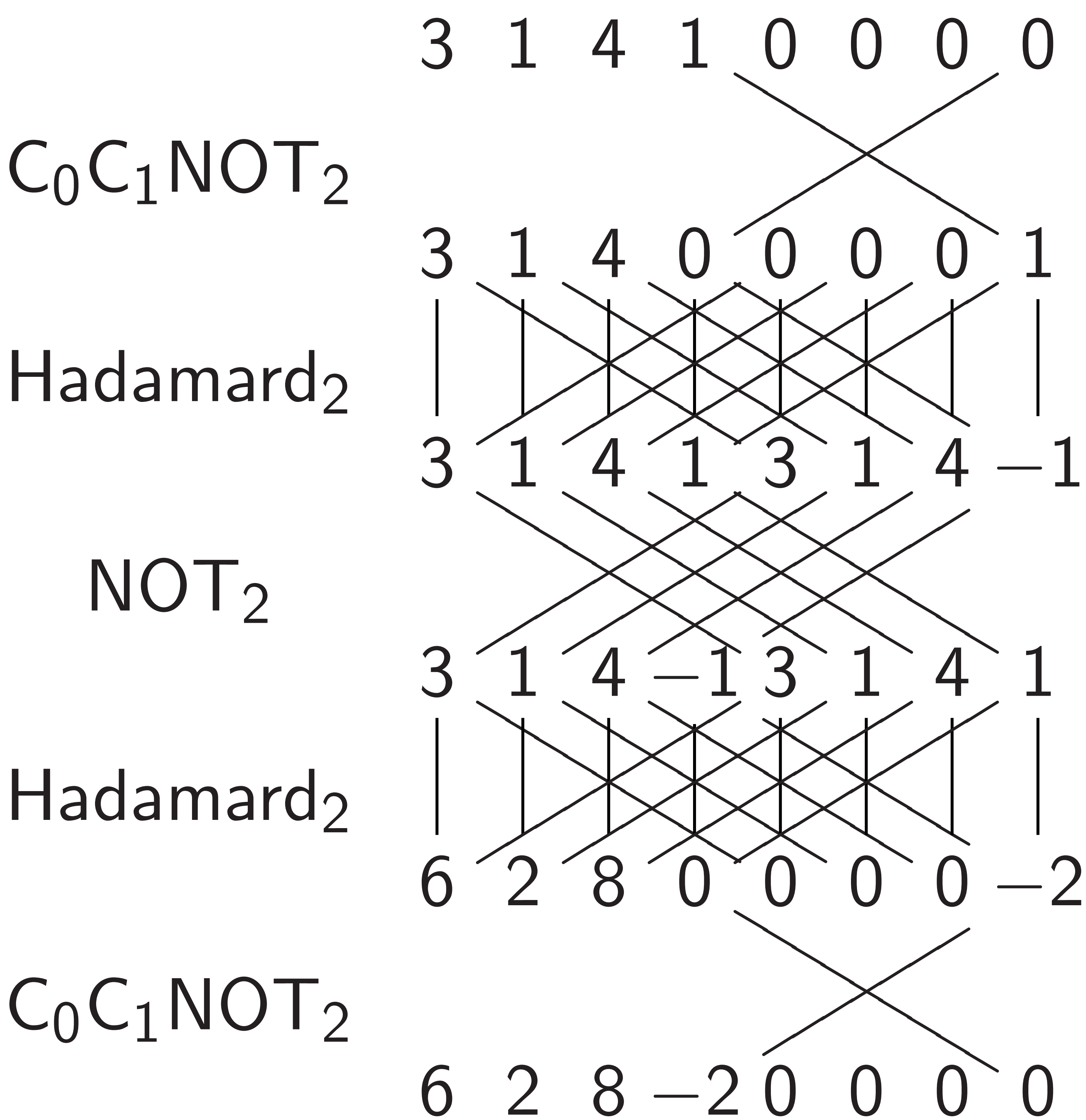
Affects measurements: "Negate amplitude around its average."

$[3, 1, 4, 1] \mapsto [1.5, 3.5, 0.5, 3.5]$.

Fancier example:

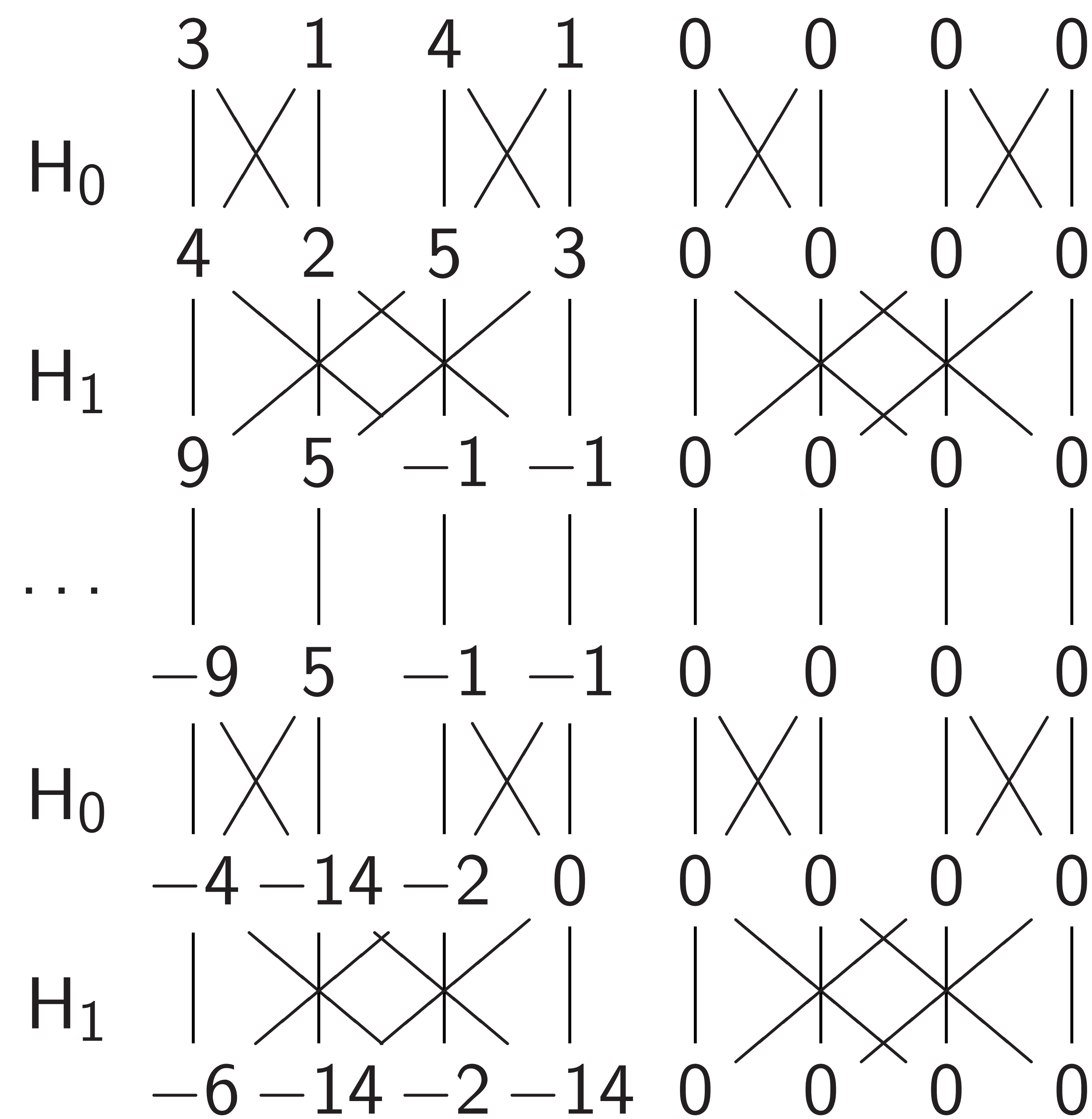
"Negate amplitude if $q_0 q_1$ is set."

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



Affects measurements: "Negate amplitude around its average."

$[3, 1, 4, 1] \mapsto [1.5, 3.5, 0.5, 3.5]$.



example:

amplitude if $q_0 q_1$ is set."

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

3 1 4 1 0 0 0 0

OT$_2$

3 1 4 0 0 0 0 1

ard$_2$

3 1 4 1 3 1 4 −1

$_2$

3 1 4 −1 3 1 4 1

ard$_2$

6 2 8 0 0 0 0 −2

OT$_2$

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

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

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

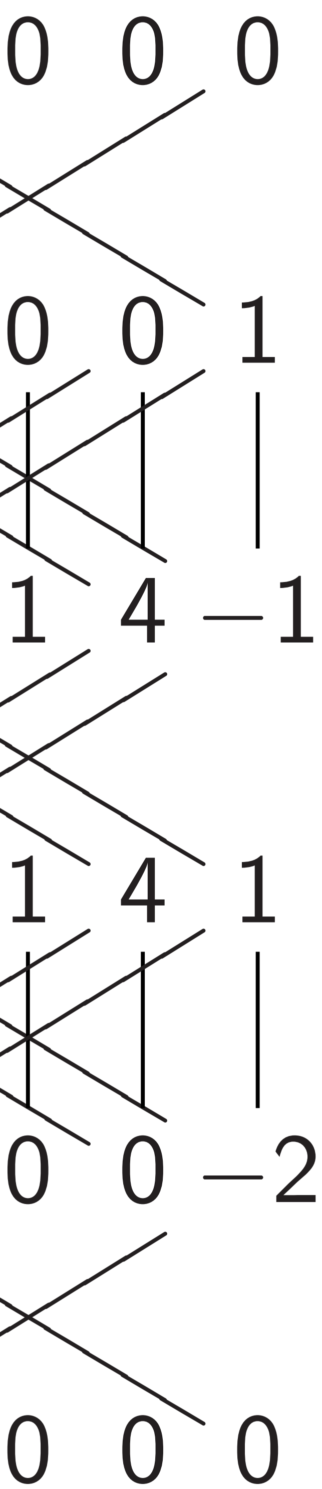
Assumptions:

- 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

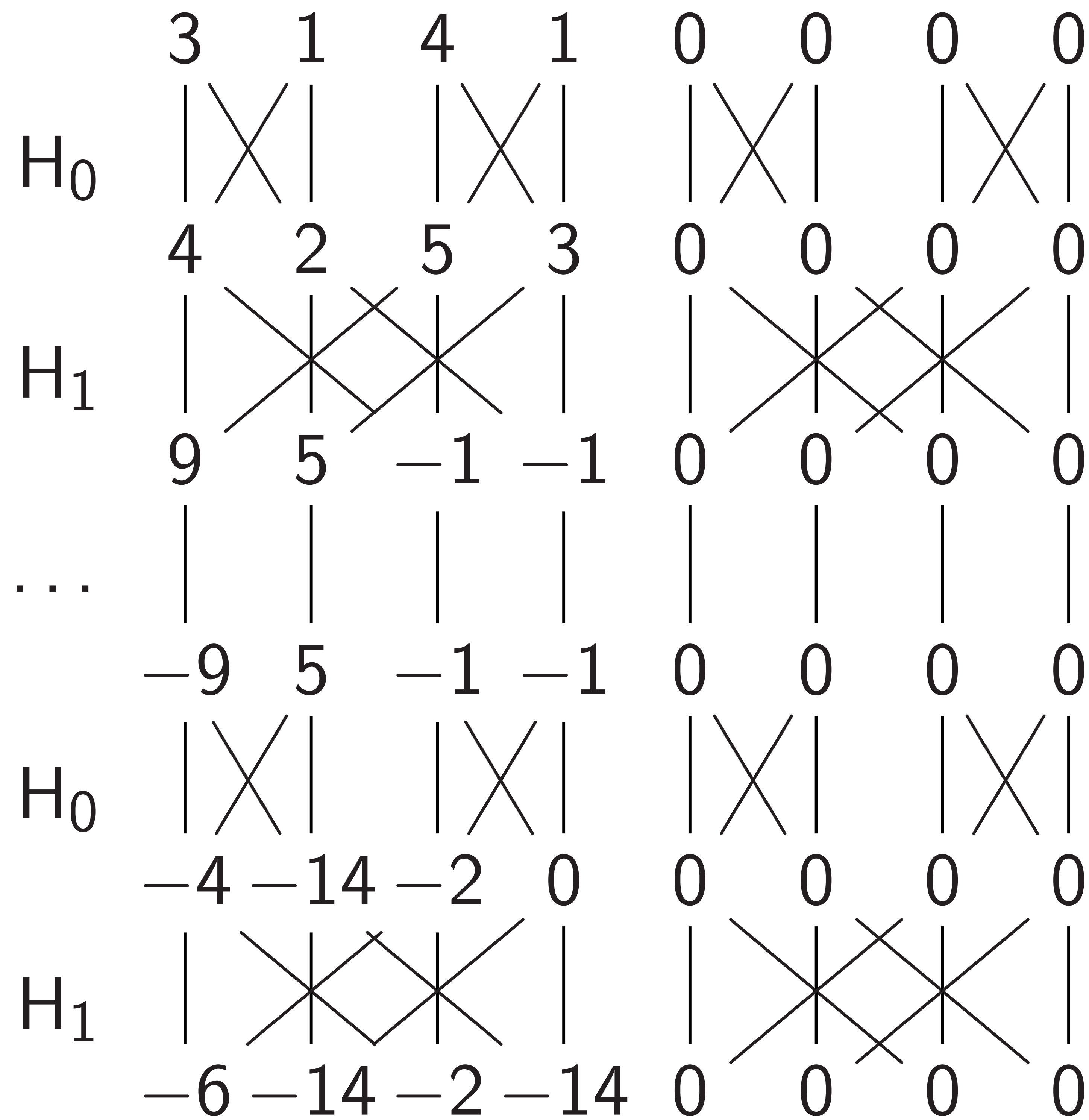
s set.”

qubit.



Affects measurements: “Negate amplitude around its average.”

$[3, 1, 4, 1] \mapsto [1.5, 3.5, 0.5, 3.5]$.



Simon’s algorithm

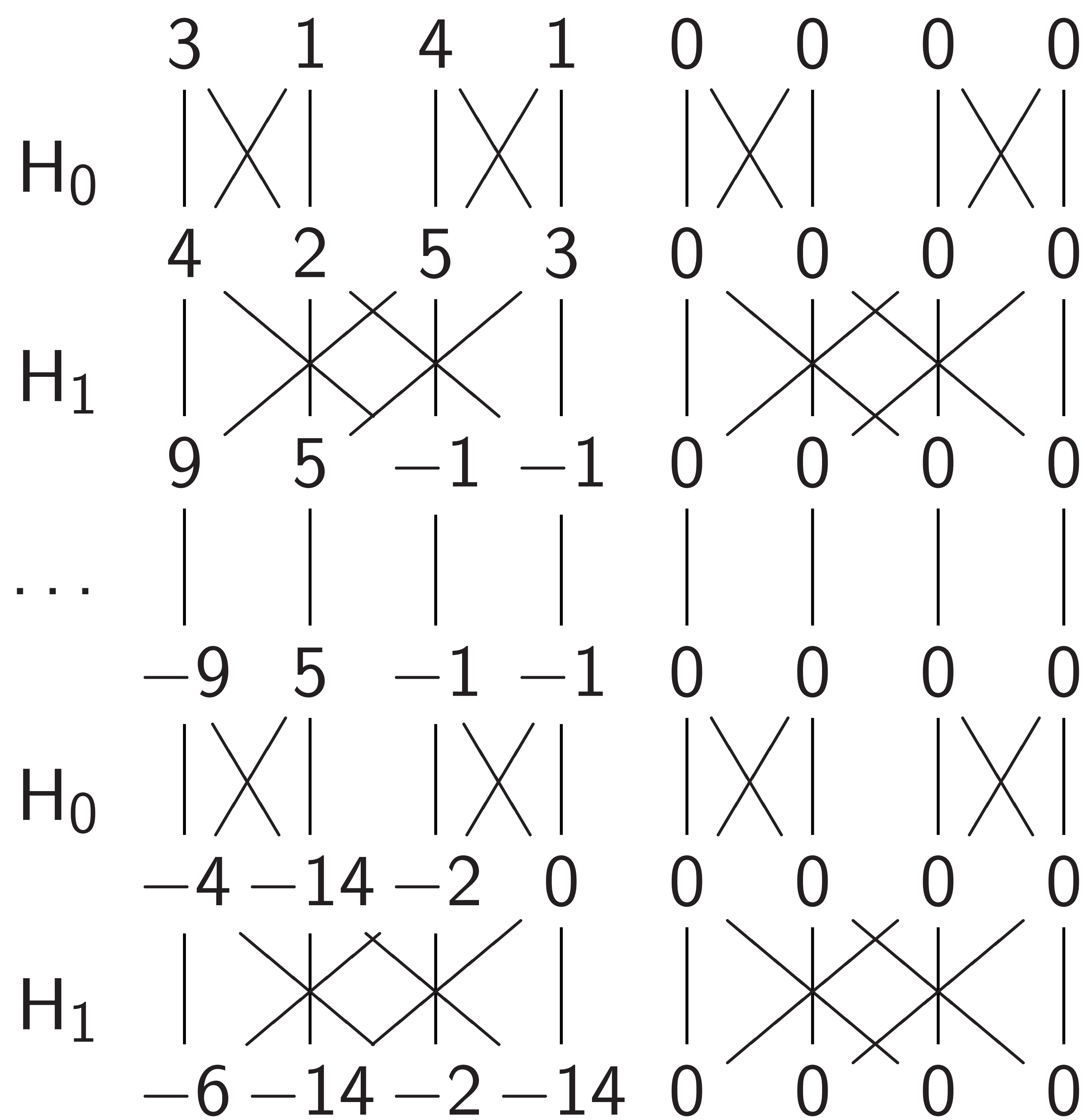
Assumptions:

- Given any $u \in \{0, 1\}^n$, can efficiently compute $f($
- 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."

$[3, 1, 4, 1] \mapsto [1.5, 3.5, 0.5, 3.5]$.



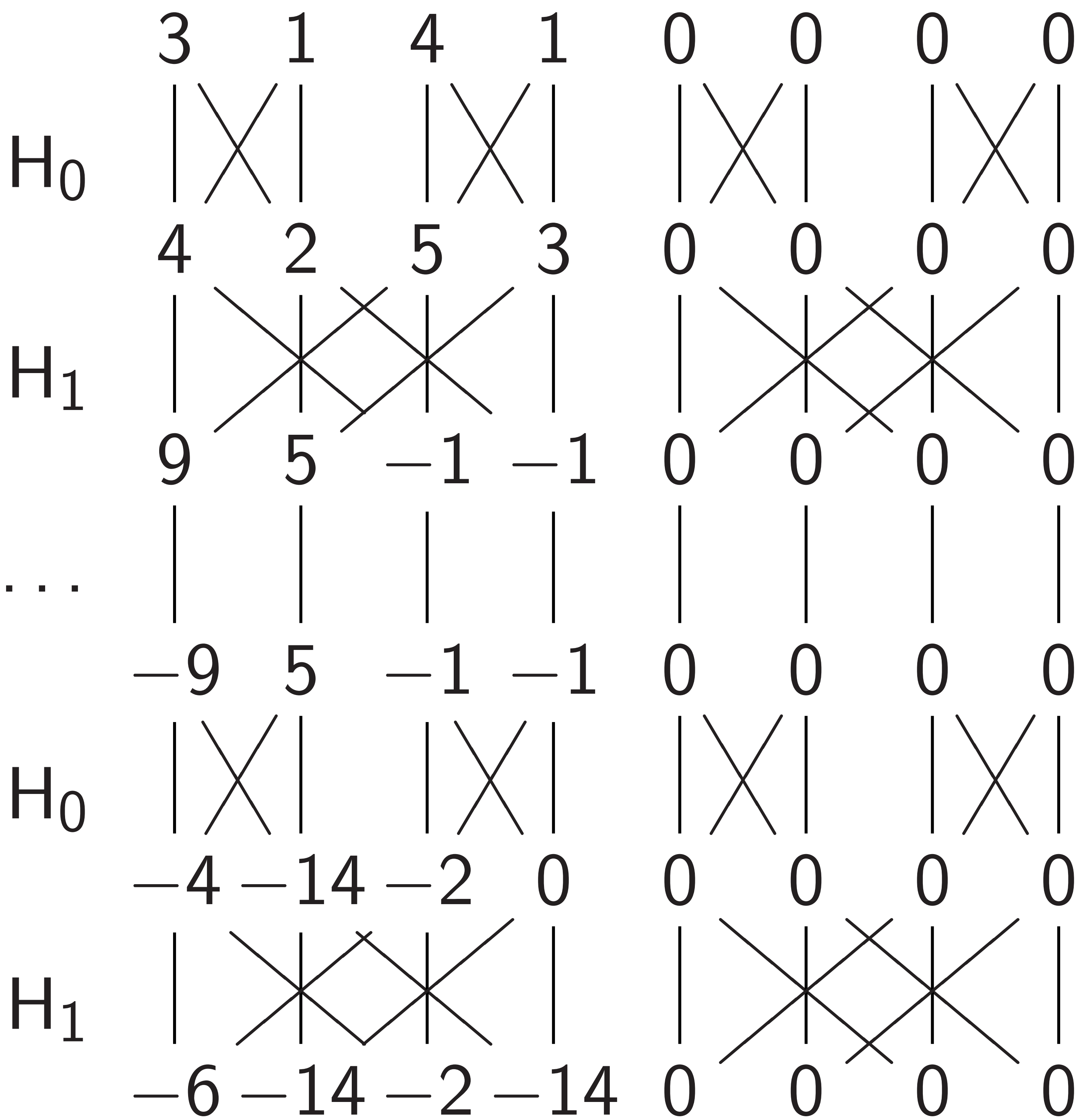
Simon's algorithm

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:

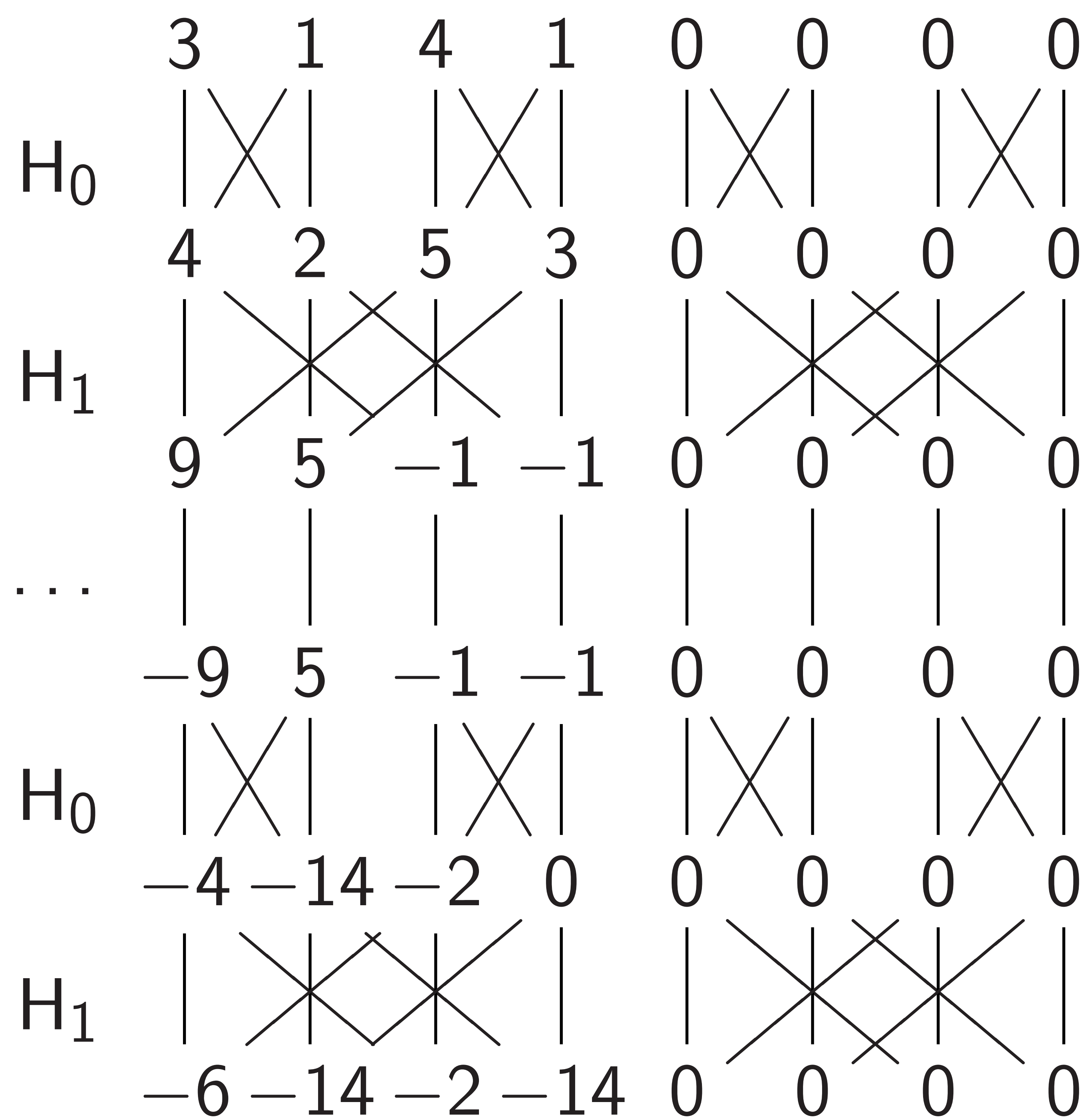
- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

Affects measurements: "Negate amplitude around its average."

$[3,1,4,1] \mapsto [1.5,3.5,0.5,3.5]$.



Simon's algorithm

Assumptions:

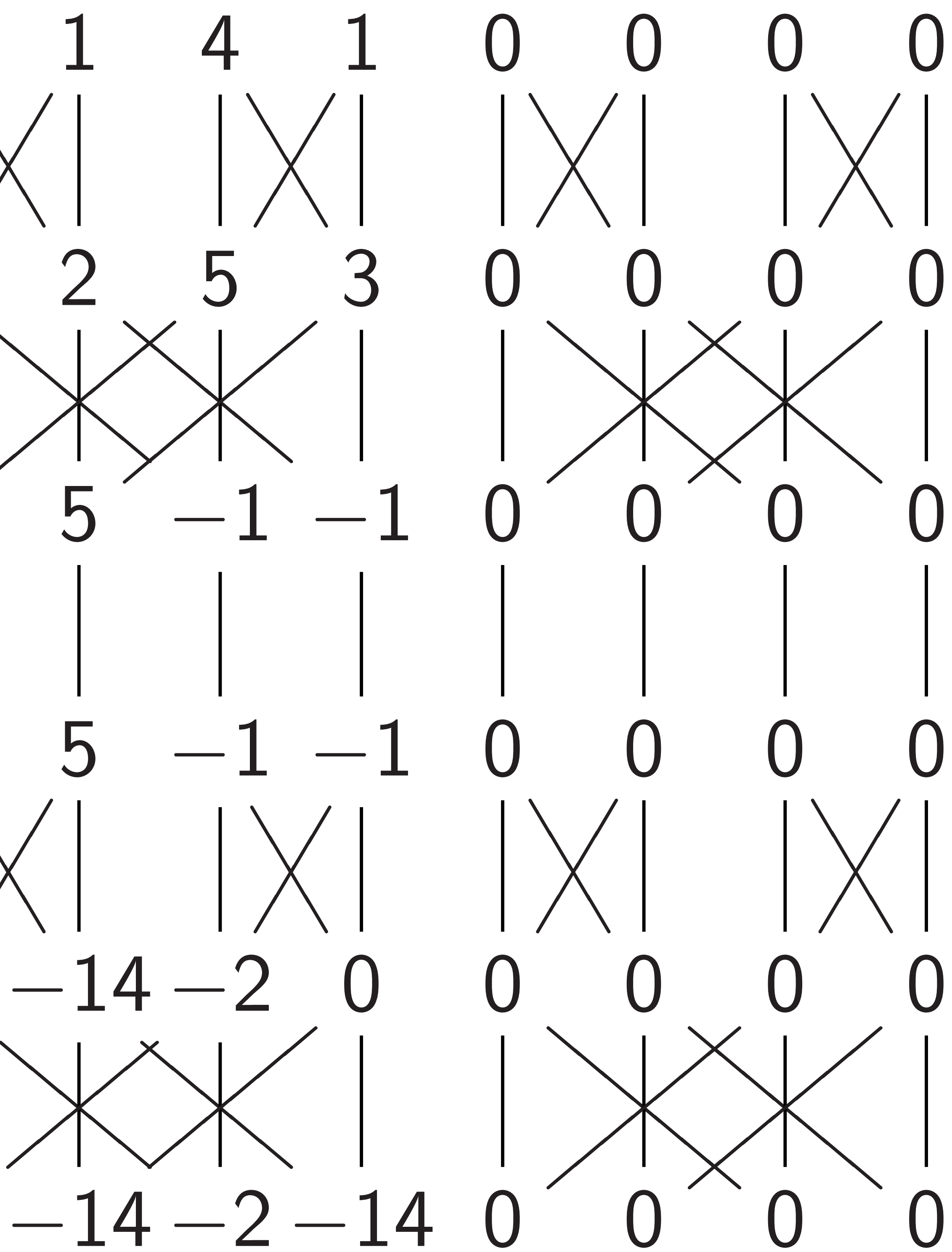
- Given any $u \in \{0,1\}^n$, can efficiently compute $f(u)$.
- Nonzero $s \in \{0,1\}^n$.
- $f(u) = f(u \oplus s)$ for all u.
- f has no other collisions.

Goal: Figure out s.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

measurements: “Negate
de around its average.”
] $\mapsto$ [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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

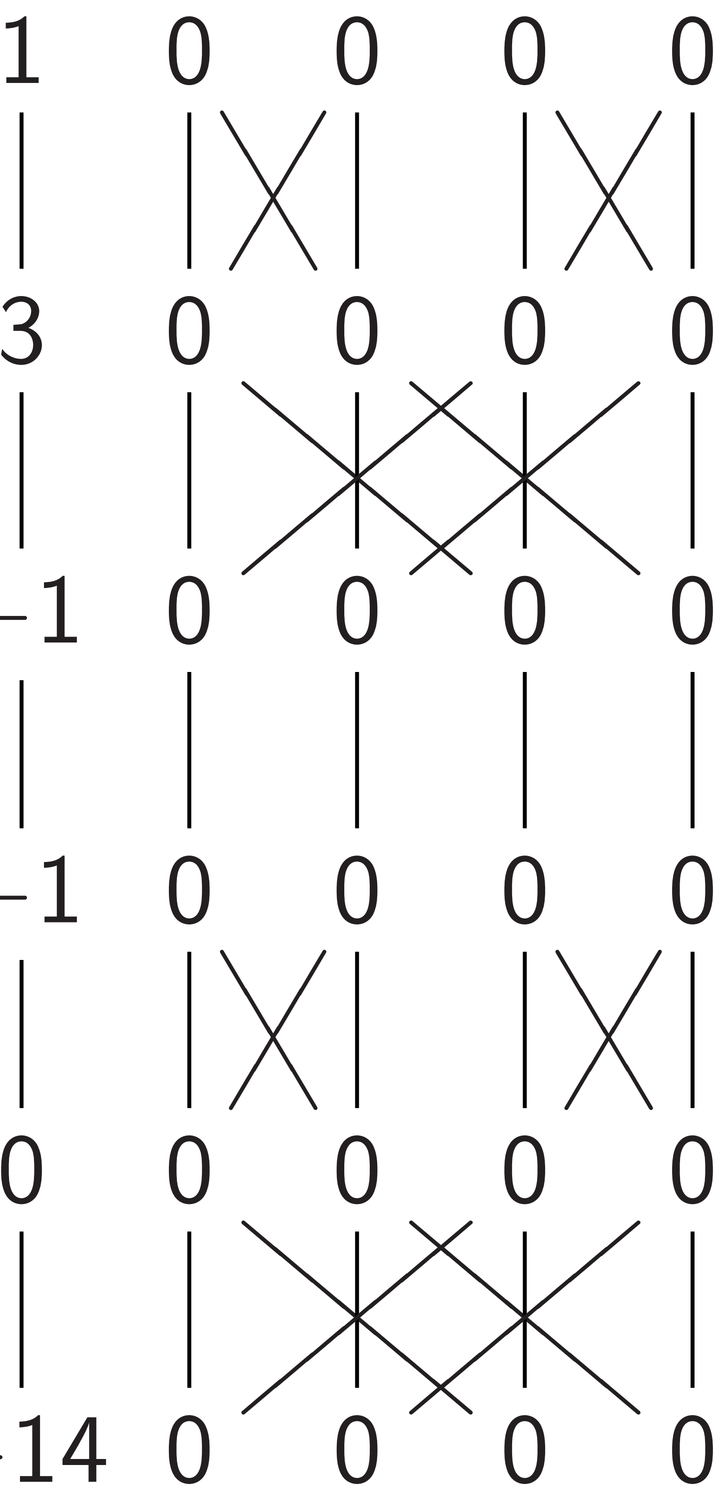
Example

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,

This exa
with 3-b

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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon

Step 1. Set up pu

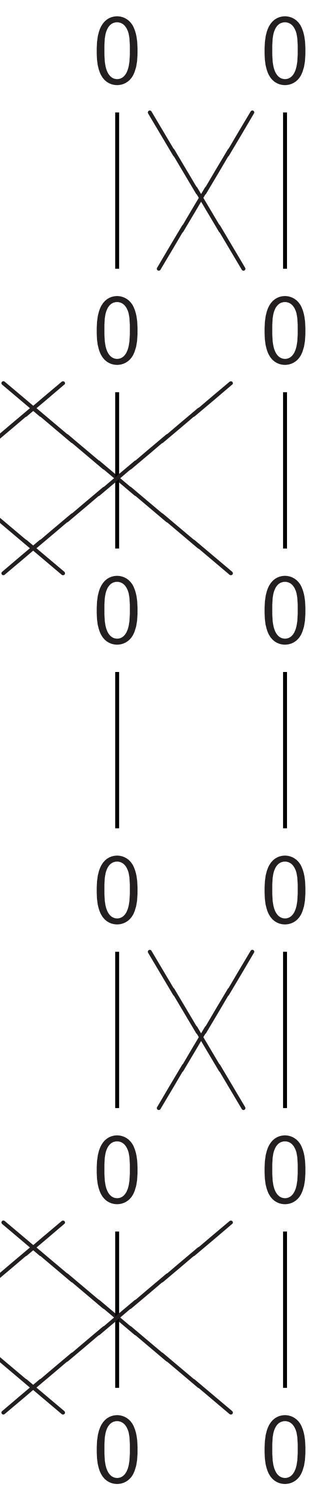
1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0,

This example is fo
with 3-bit input ar

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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

Simon’s algorithm finds s with $\approx n$ quantum evaluations 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.

This example is for a functio

with 3-bit input and 3-bit ou

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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

Simon's algorithm finds s with $\approx n$ quantum evaluations 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.

This example is for a function f with 3-bit input and 3-bit output.

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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 2.0. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 2.1. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 2.2. 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. Step 3 will apply the function f (a specific function in this example), computing $f(u)$ in universe u.

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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3a. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3b. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3c. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3d. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3e. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3f. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3g. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3h. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3i. More entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3j. Final entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 3j. Final entry 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 4.0. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 4.1. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 4.2. 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.

Non-quantum algorithm to find s: compute f for many inputs, hope to find collision.

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

Example of Simon's algorithm

Step 4.2. 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 5: 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.

ntum algorithm to find s:

e f for many inputs,

find collision.

algorithm finds s with

ntum evaluations of f.

Example of Simon's algorithm

Step 4.2. 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 5: 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.

orithm to find s:
ny inputs,
ion.

finds s with
uations of f.

Example of Simon's algorithm

Step 4.2. Hadamard$_2$:

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

Step 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure o

u).

find s:

ith

f.

Example of Simon's algorithm

Step 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Example of Simon's algorithm

Step 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Example of Simon's algorithm

Step 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 3 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 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 3 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 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 3 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 4.2. 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 5: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.

Repeat to figure out 101.

Generalize Step 3 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

2. 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 3 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 $f(s)$

Goal: Fi

Non-qua
compute
hope to
Success
until #t

's algorithm

rd$_2$:

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 3 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 $f(s) = 0$.

Goal: Figure out s

Non-quantum algo
compute f for ma
hope to find outpu
Success probability
until #tries appro

m

ome

se: a

o 101.

Repeat to figure out 101.

Generalize Step 3 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 $f(s) = 0$.

Goal: Figure out s.

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

Repeat to figure out 101.

Generalize Step 3 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 $f(s) = 0$.

Goal: Figure out s.

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

Repeat to figure out 101.

Generalize Step 3 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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$ quantum evaluations of f. e.g. 2^{64} instead of 2^{128}.

to figure out 101.

ze Step 3 to any function
) with $f(u) = f(u \oplus s)$.
" algorithm figures out s.

lgorithm replaces $\oplus$
re general $+$ operation.
ectacular 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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$
quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start fro
over n-b

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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$
quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start from uniform
over n-bit strings

nction
$\oplus\ s)$.
out s.

ion.
ns.

with

t with
od p.
gs.

Grover's algorithm

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

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$
quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start from uniform superpos
over n-bit strings u: each a_u

Grover's algorithm

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

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$ quantum evaluations of f. e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

Grover's algorithm

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

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$ quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

Grover's algorithm

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

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$
quantum evaluations of f.
e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition
over n-bit strings u: each $a_u = 1$.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f 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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$ quantum evaluations of f. e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f 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 $f(s) = 0$.

Goal: Figure out s.

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

Grover's algorithm takes only $2^{n/2}$ quantum evaluations of f. e.g. 2^{64} instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f 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 $s \in \{0,1\}^n$
$= 0$.

gure out s.

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

algorithm takes only $2^{n/2}$
n evaluations of f.
instead of 2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

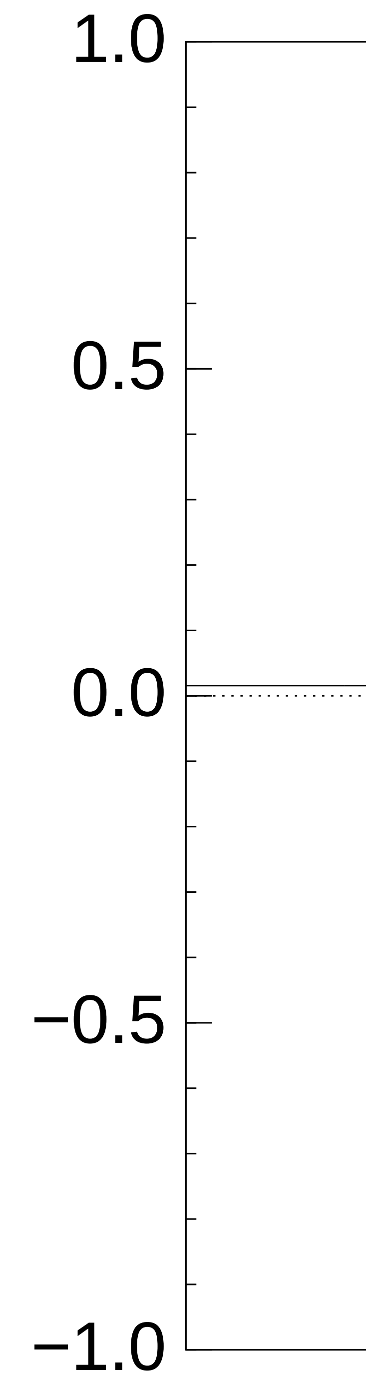
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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

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

s.

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

takes only $2^{n/2}$
ons of f.
2^{128}.

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

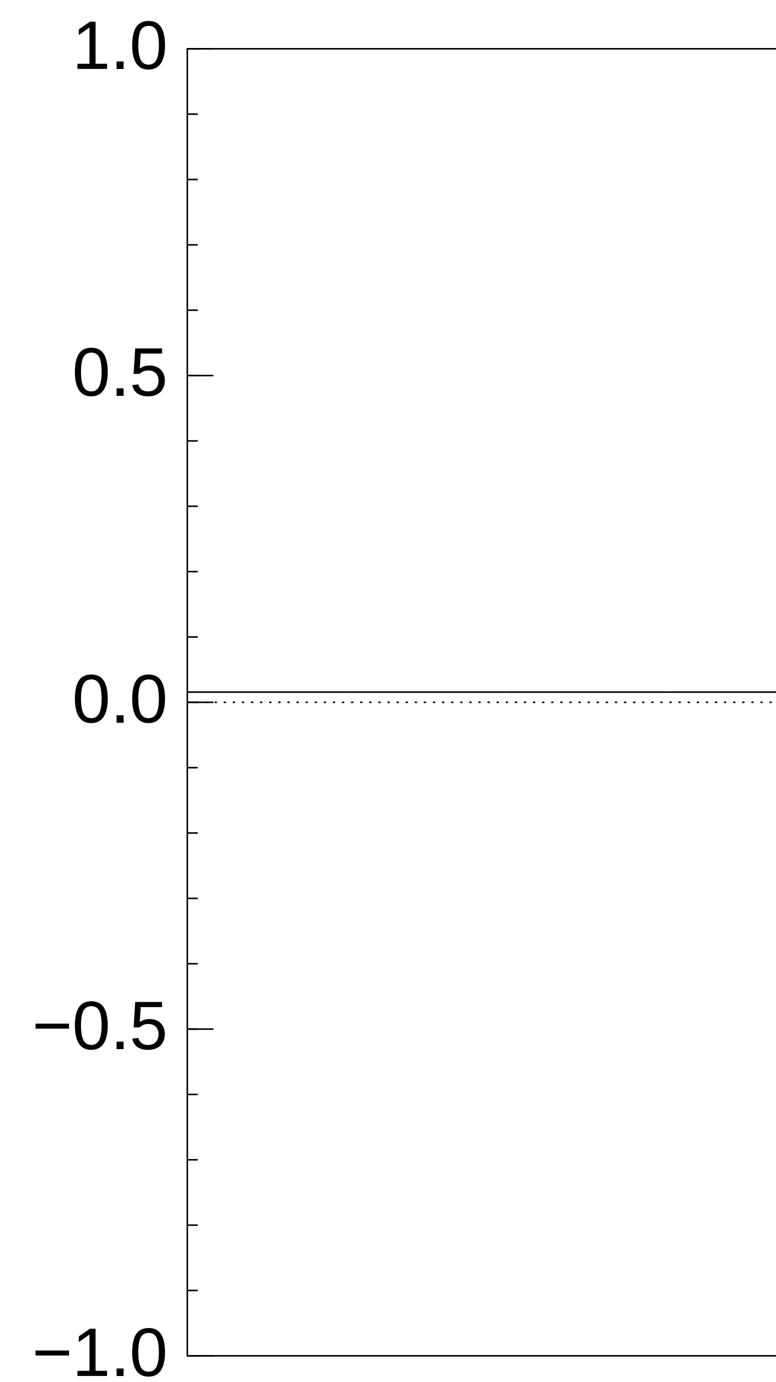
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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:



find s:

ow

y $2^{n/2}$

Start from uniform superposition over n-bit strings u: each $a_u = 1$.

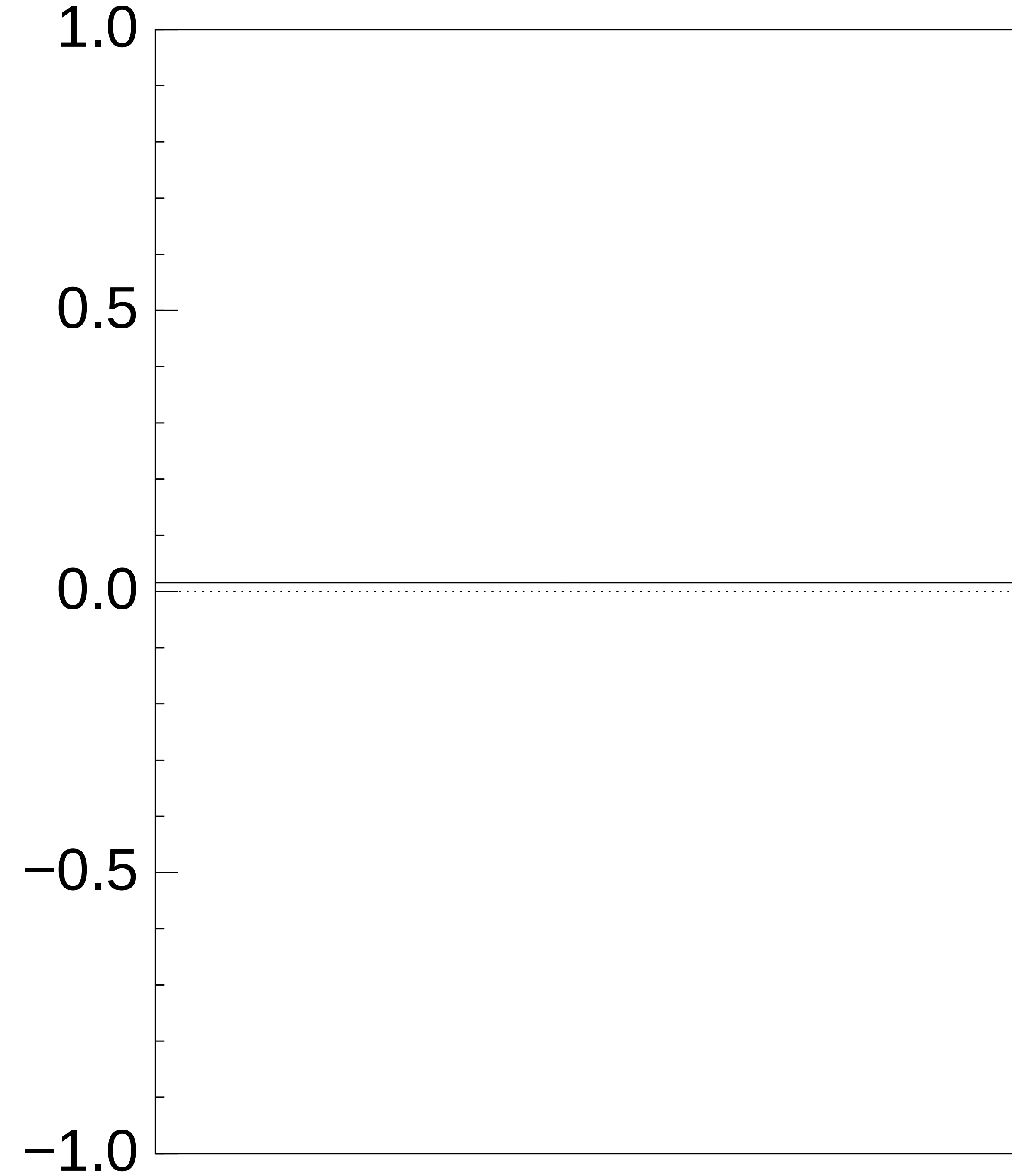
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

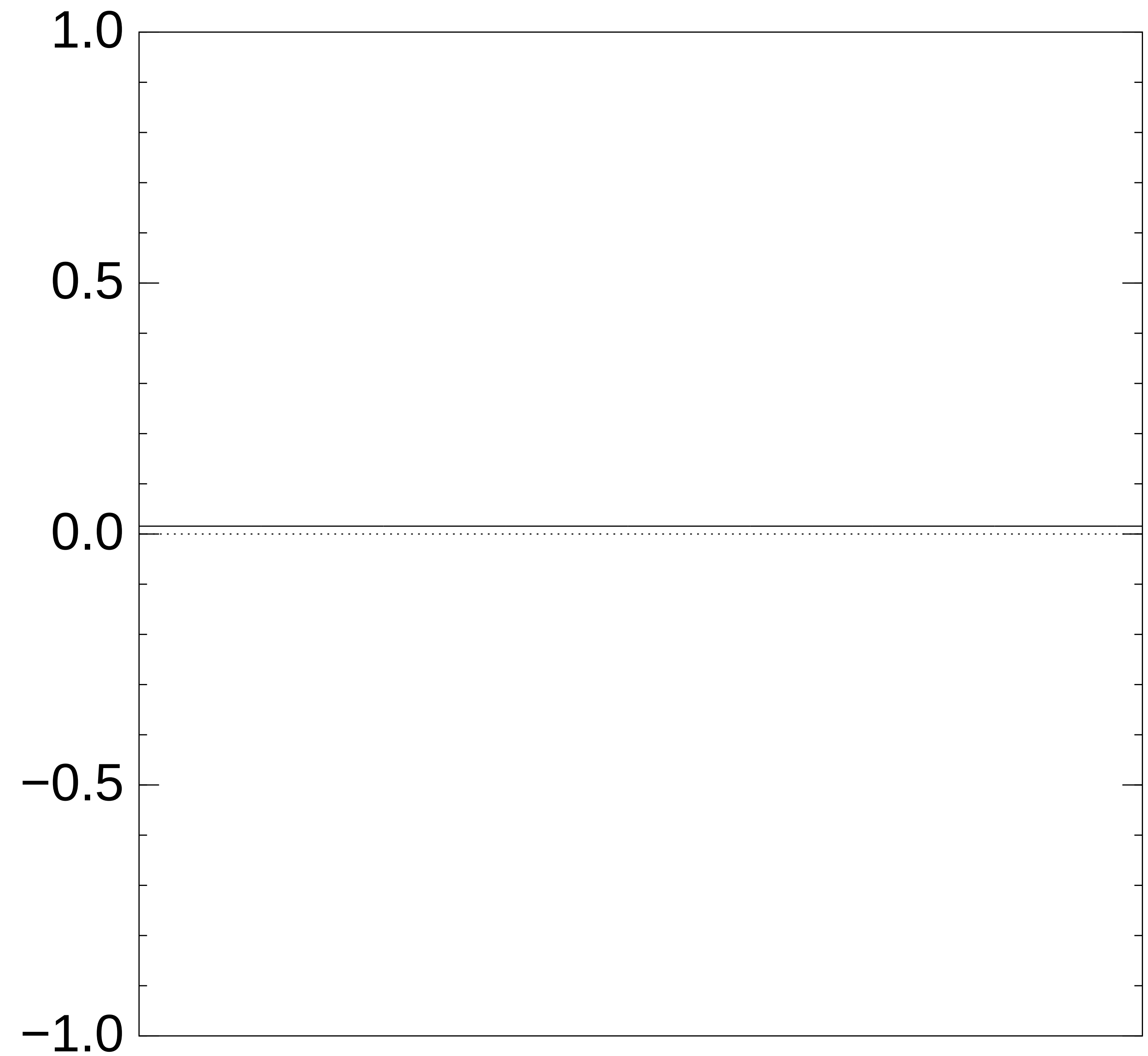
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

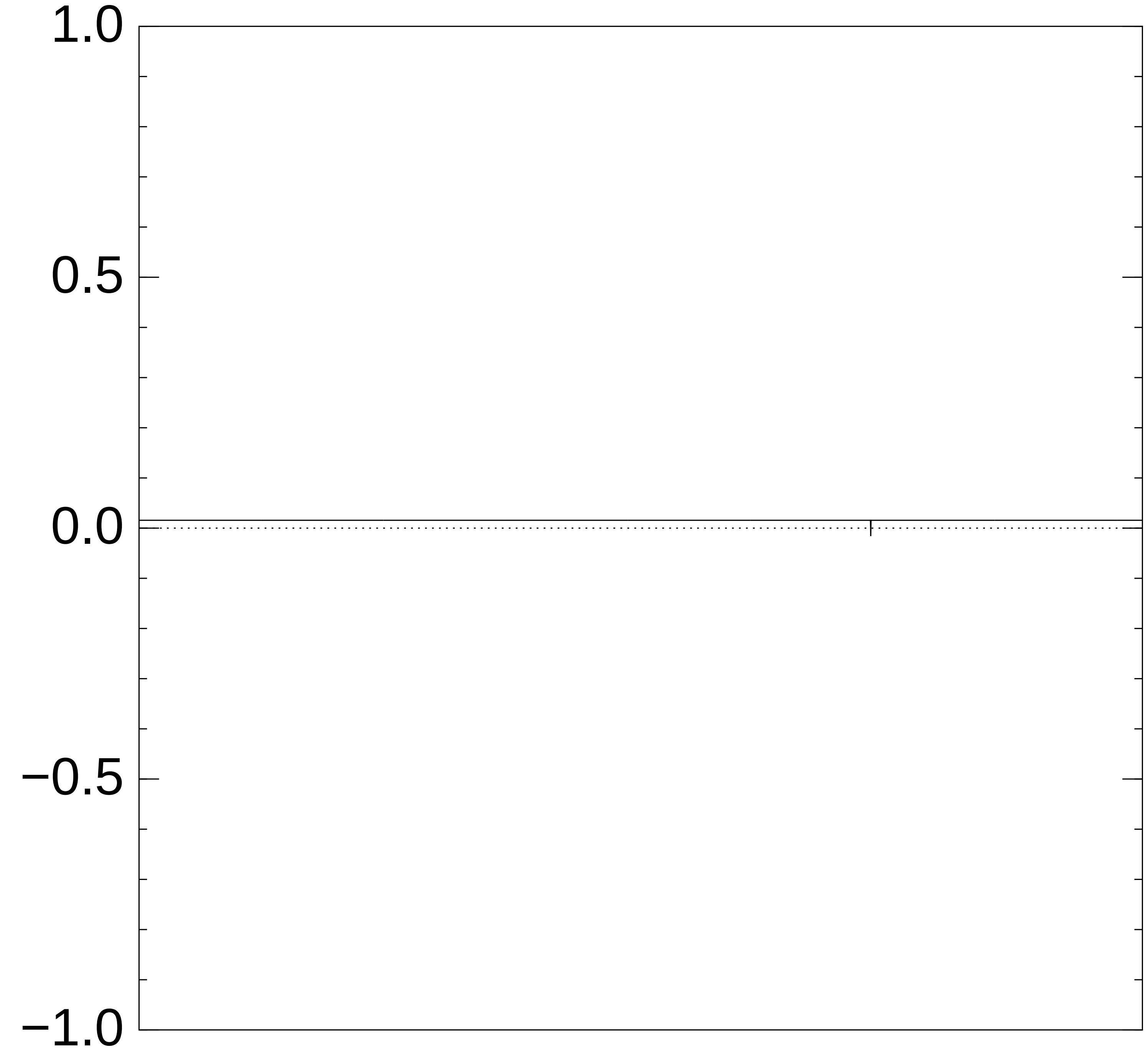
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

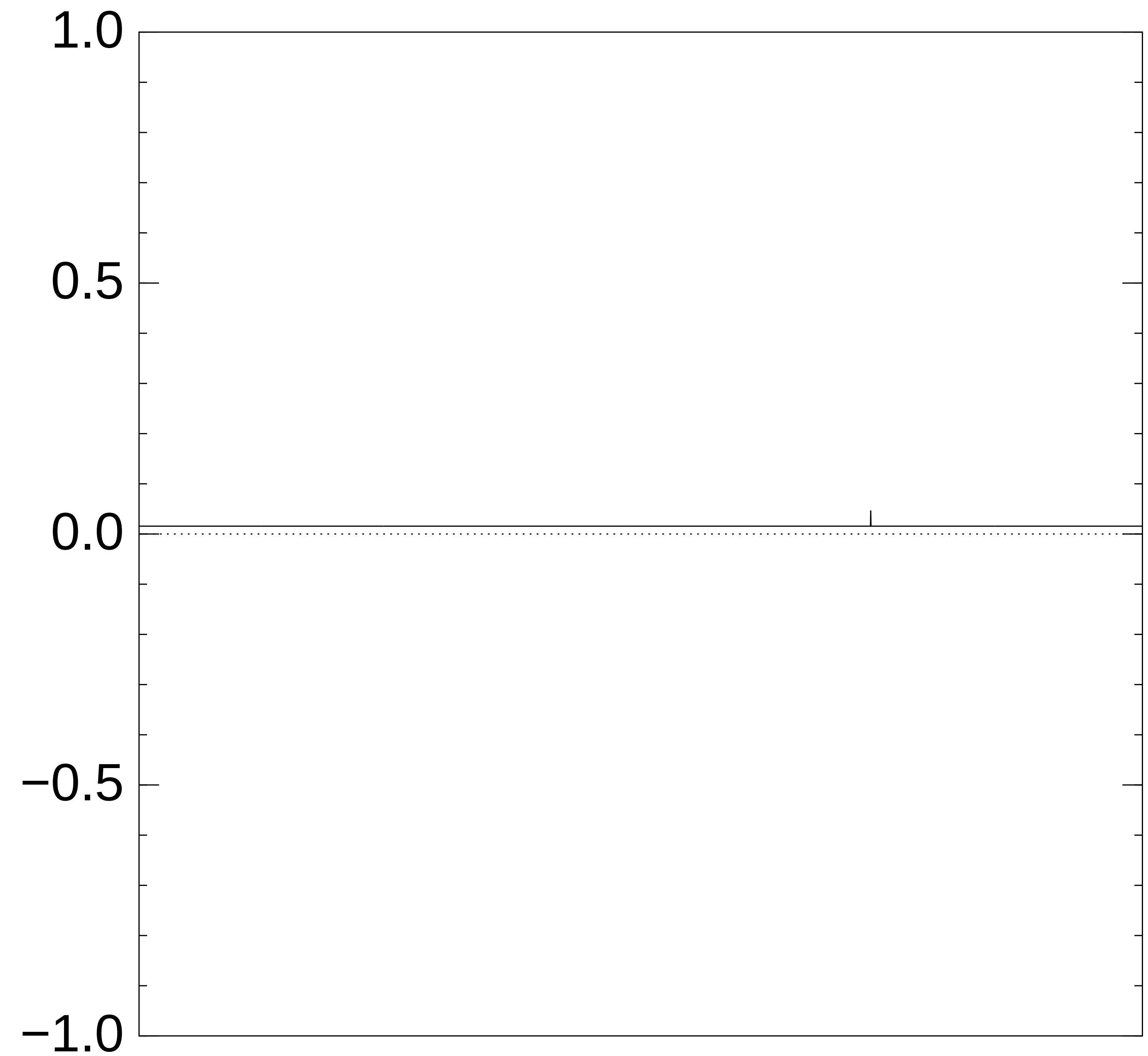
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

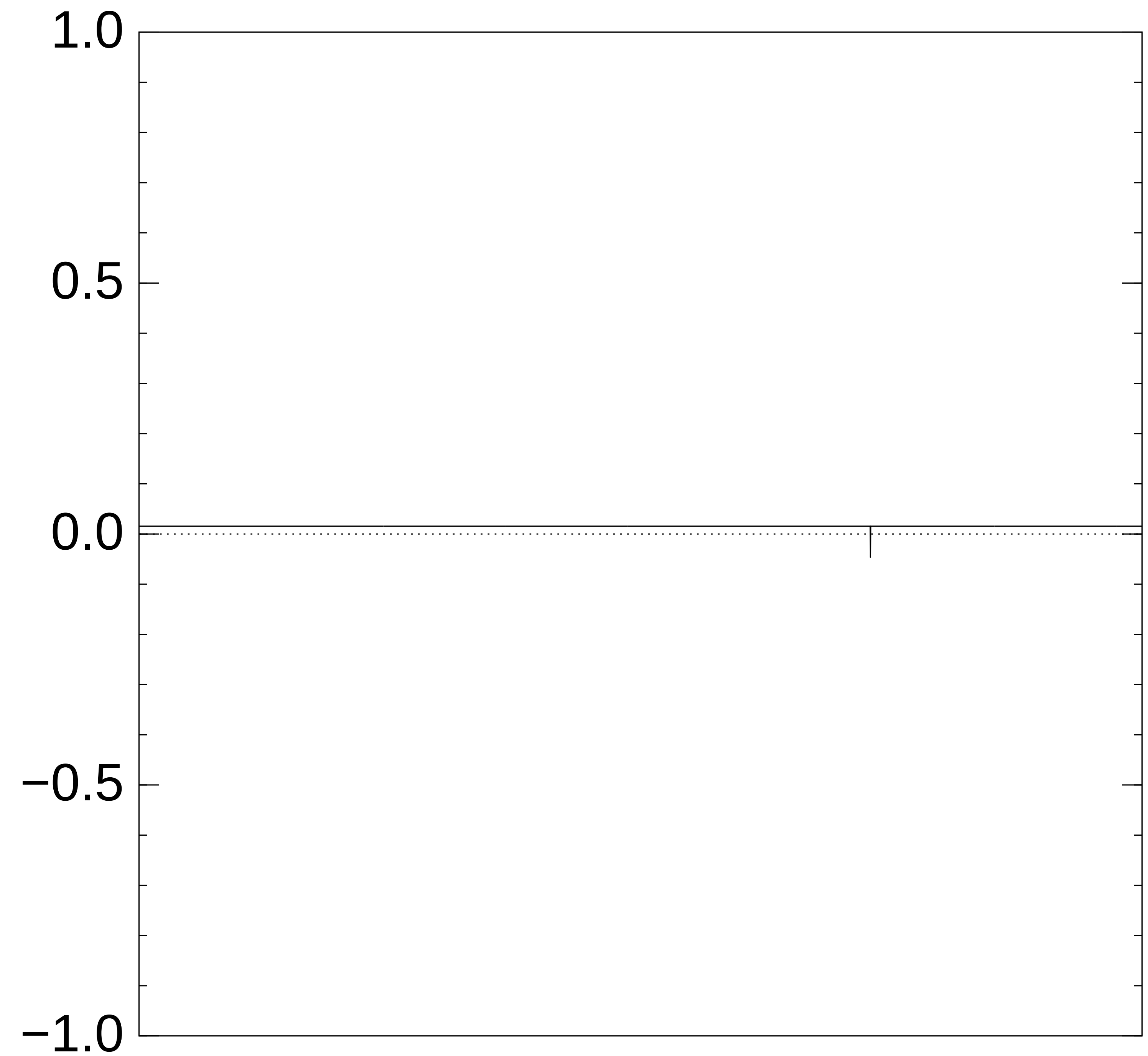
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

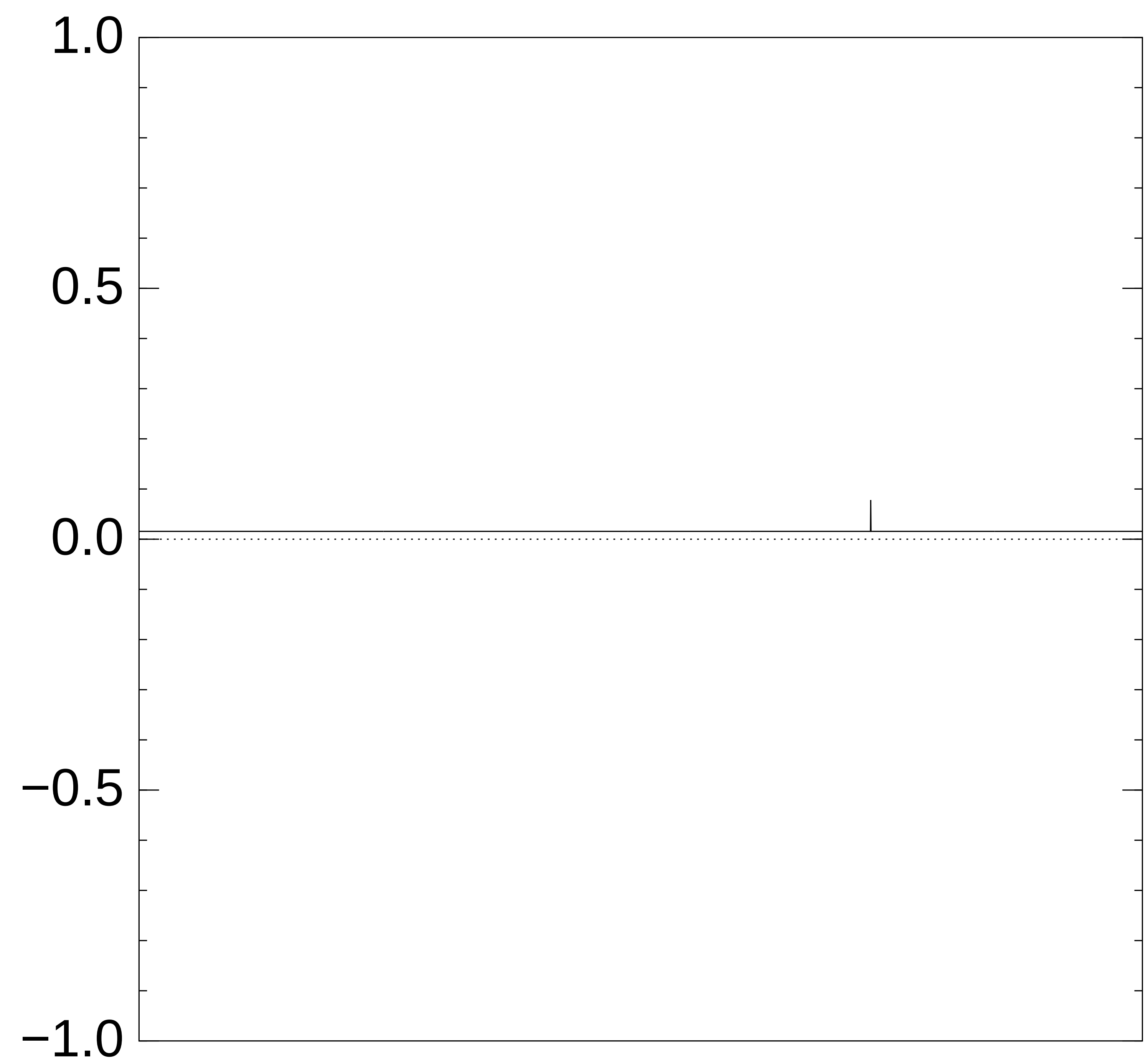
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

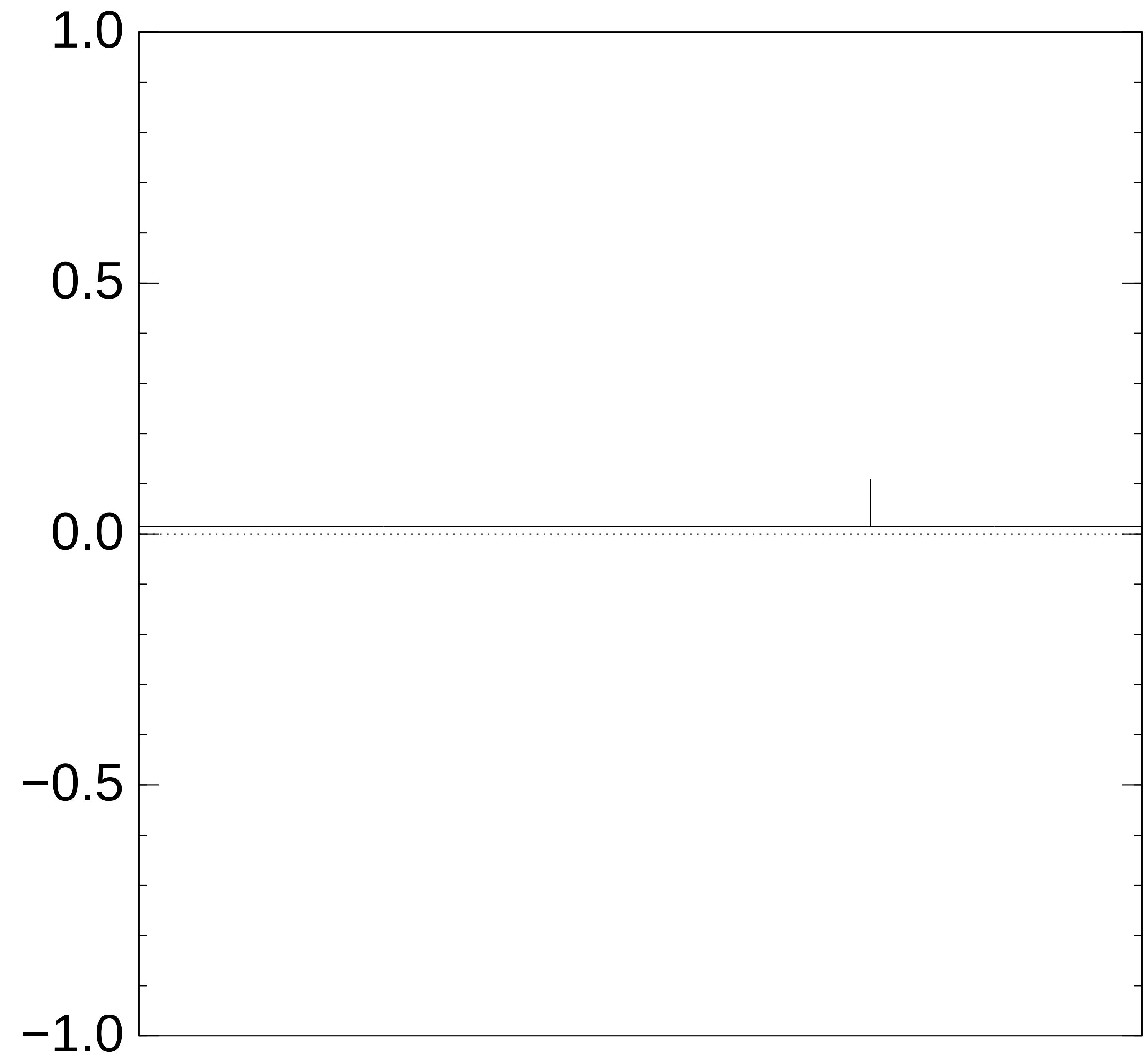
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 $3 \times (\text{Step } 1 + \text{Step } 2)$:



Start from uniform superposition over n-bit strings u: each $a_u = 1$.

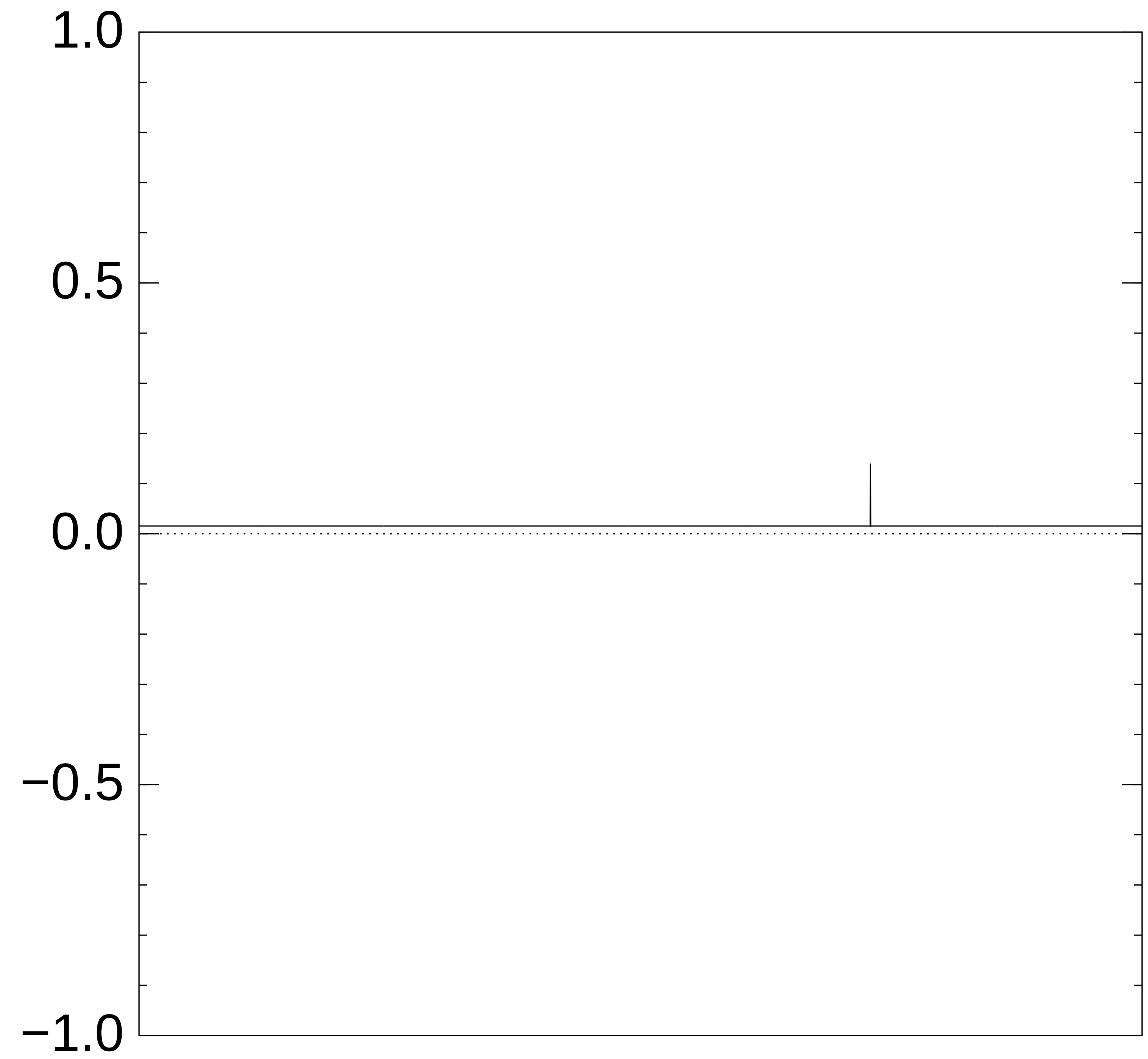
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

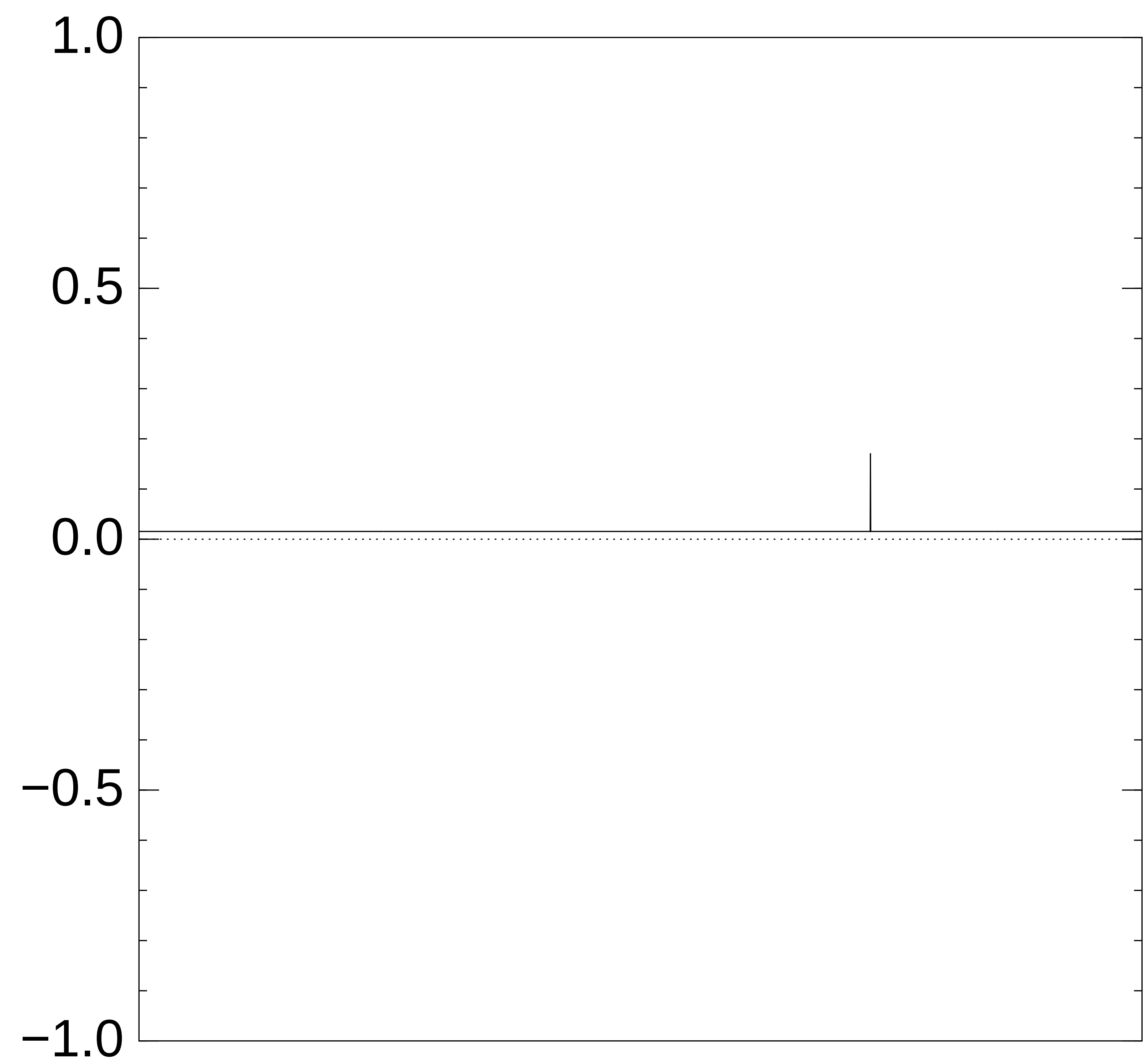
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

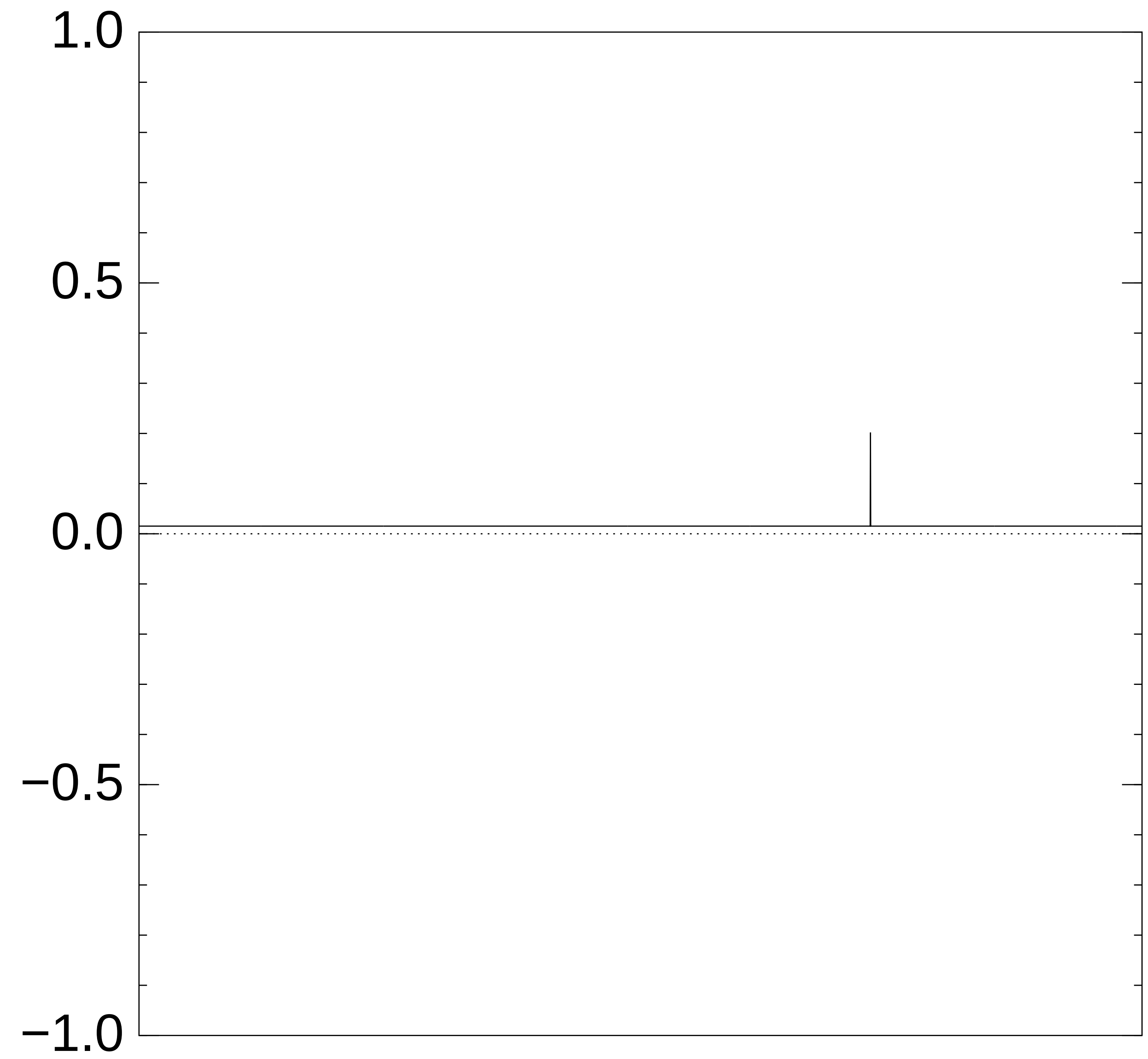
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 $6 \times$ (Step 1 + Step 2):



Start from uniform superposition over n-bit strings u: each $a_u = 1$.

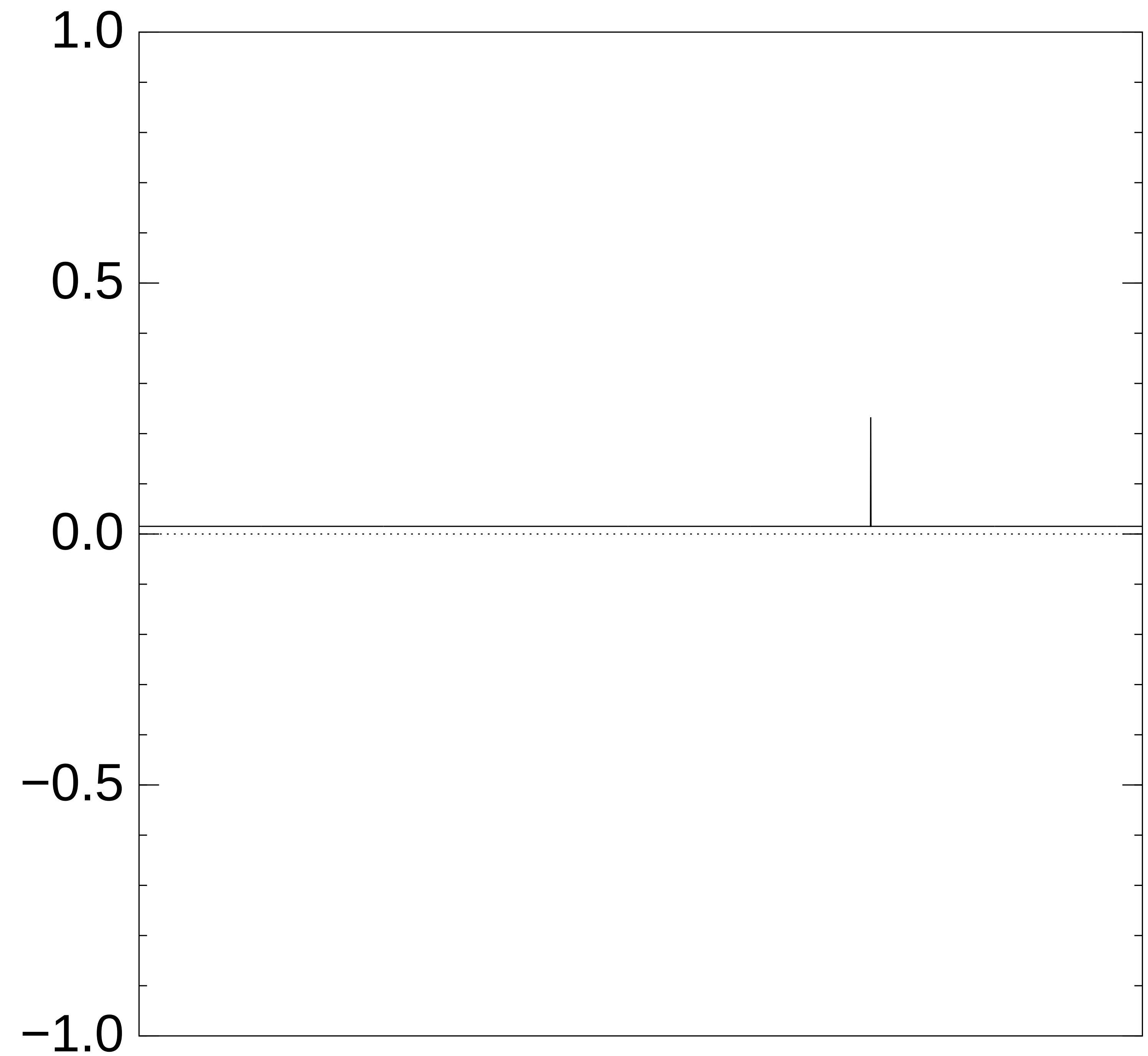
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

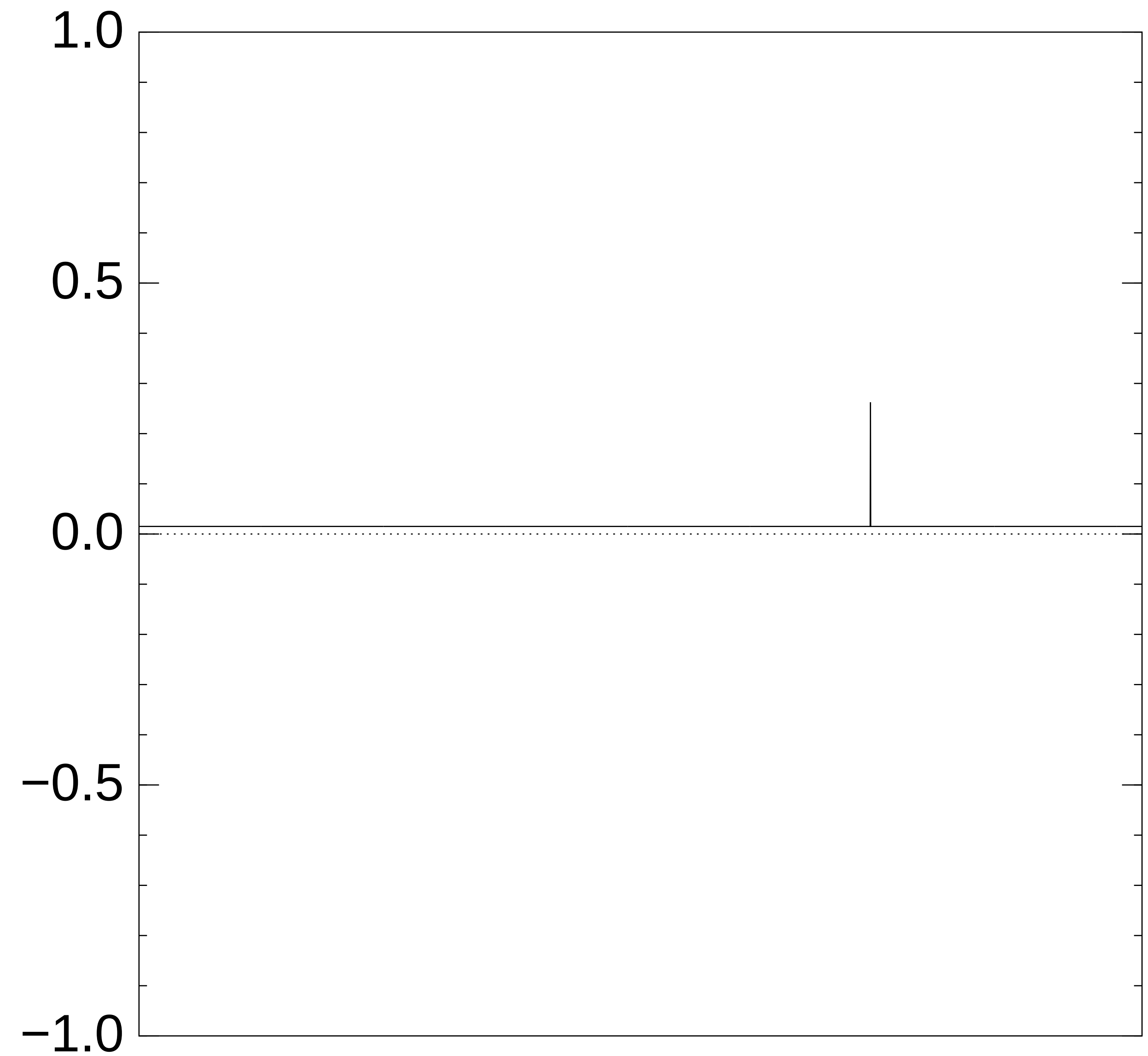
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

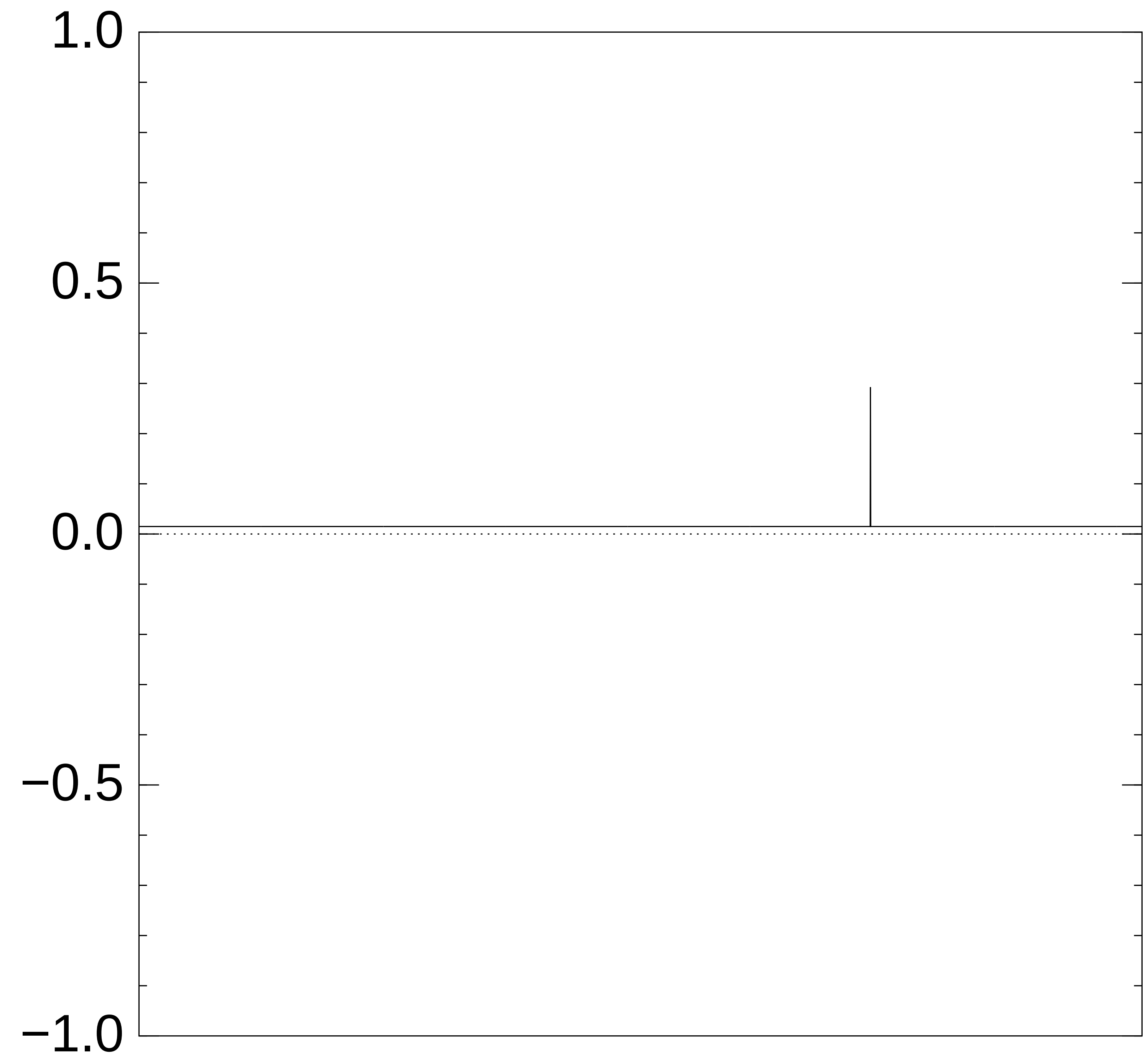
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

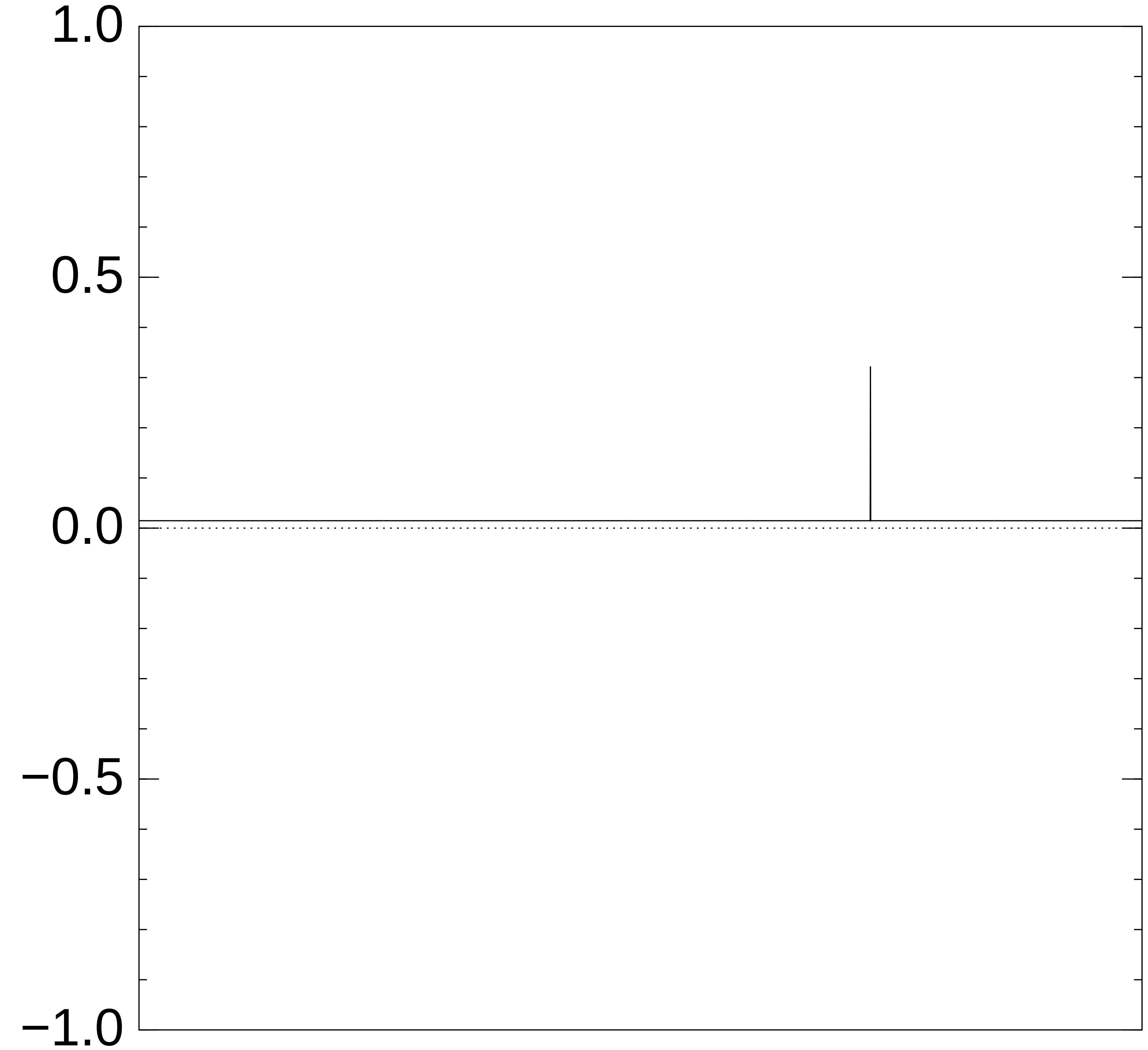
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

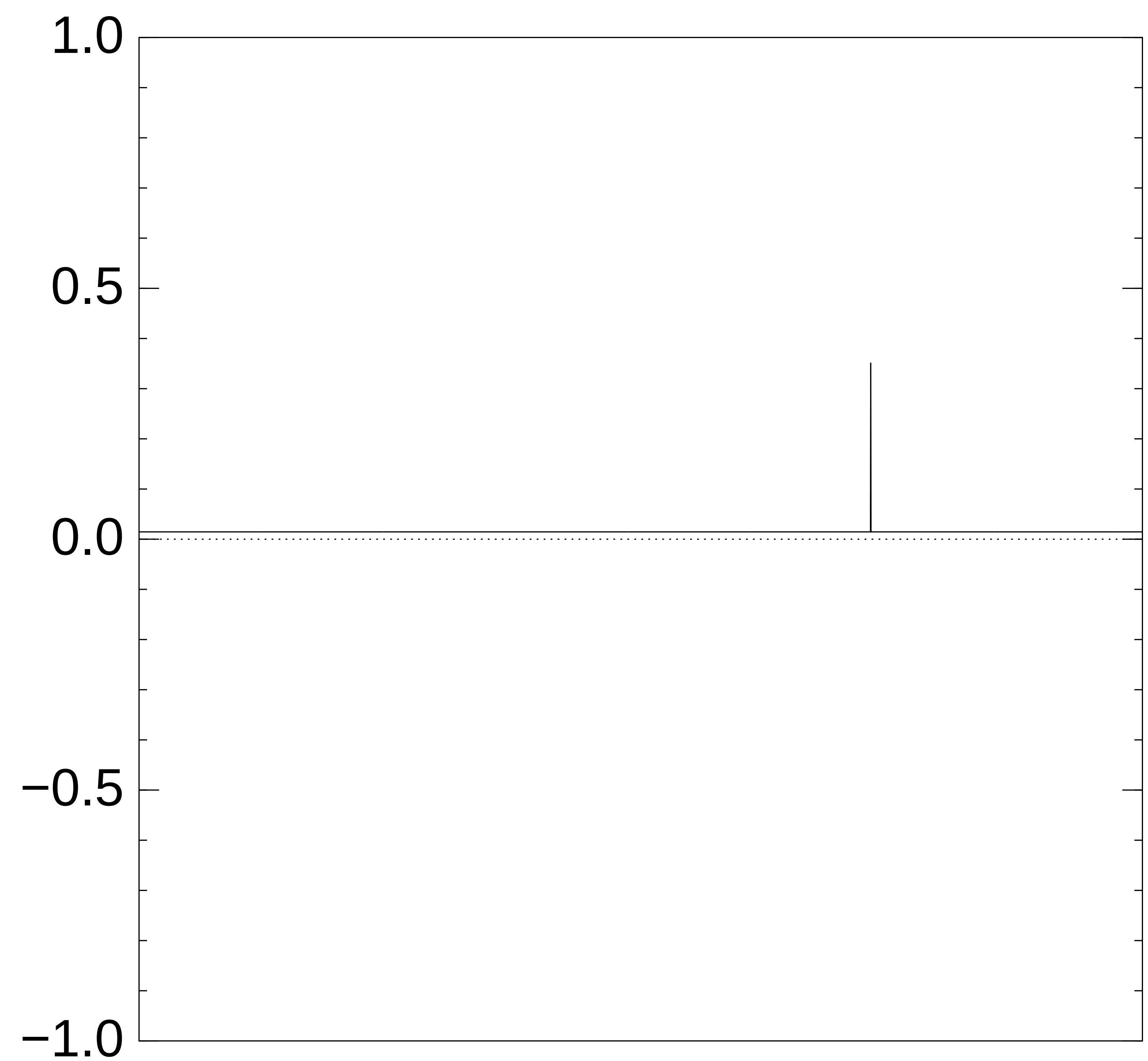
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

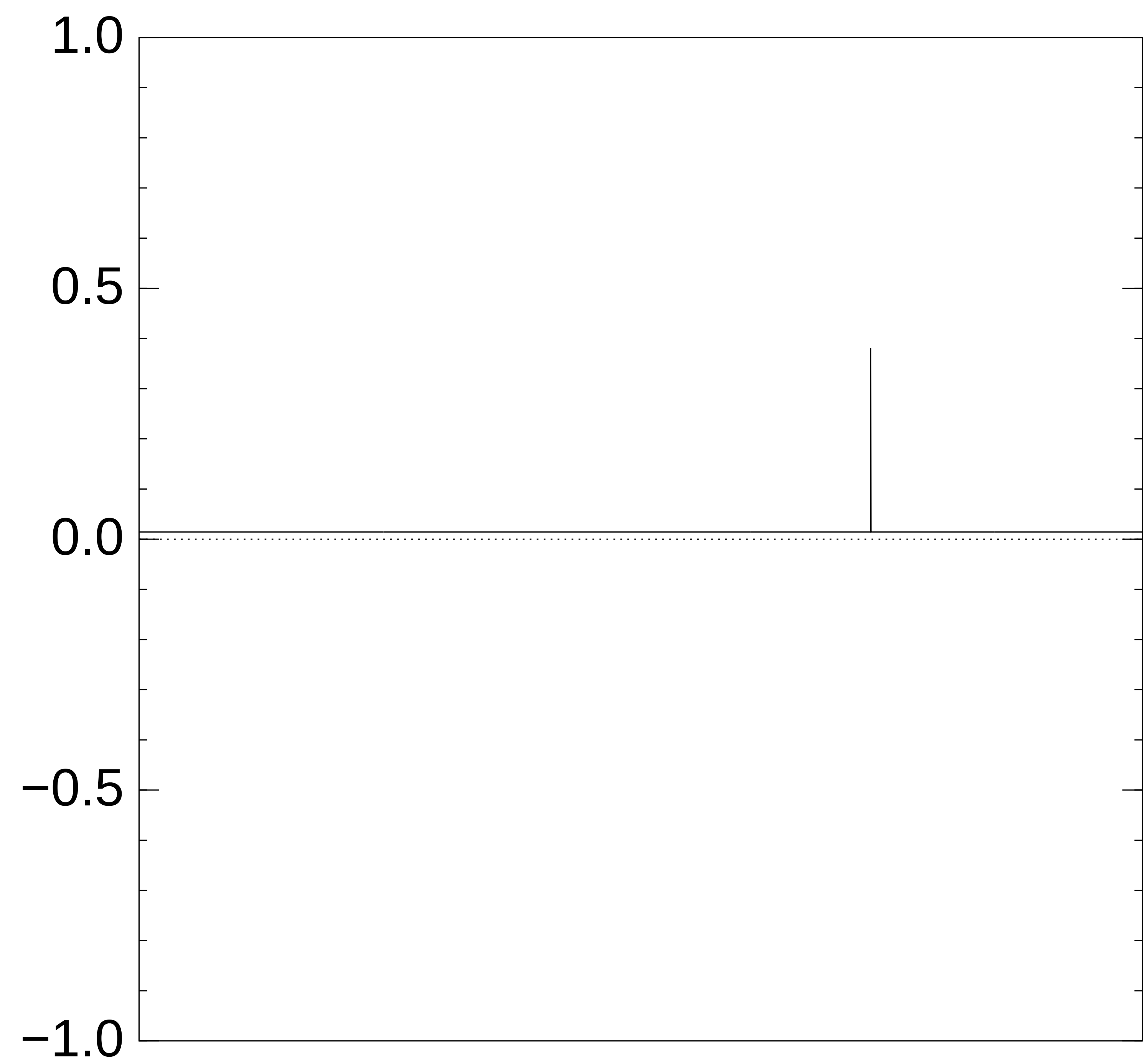
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

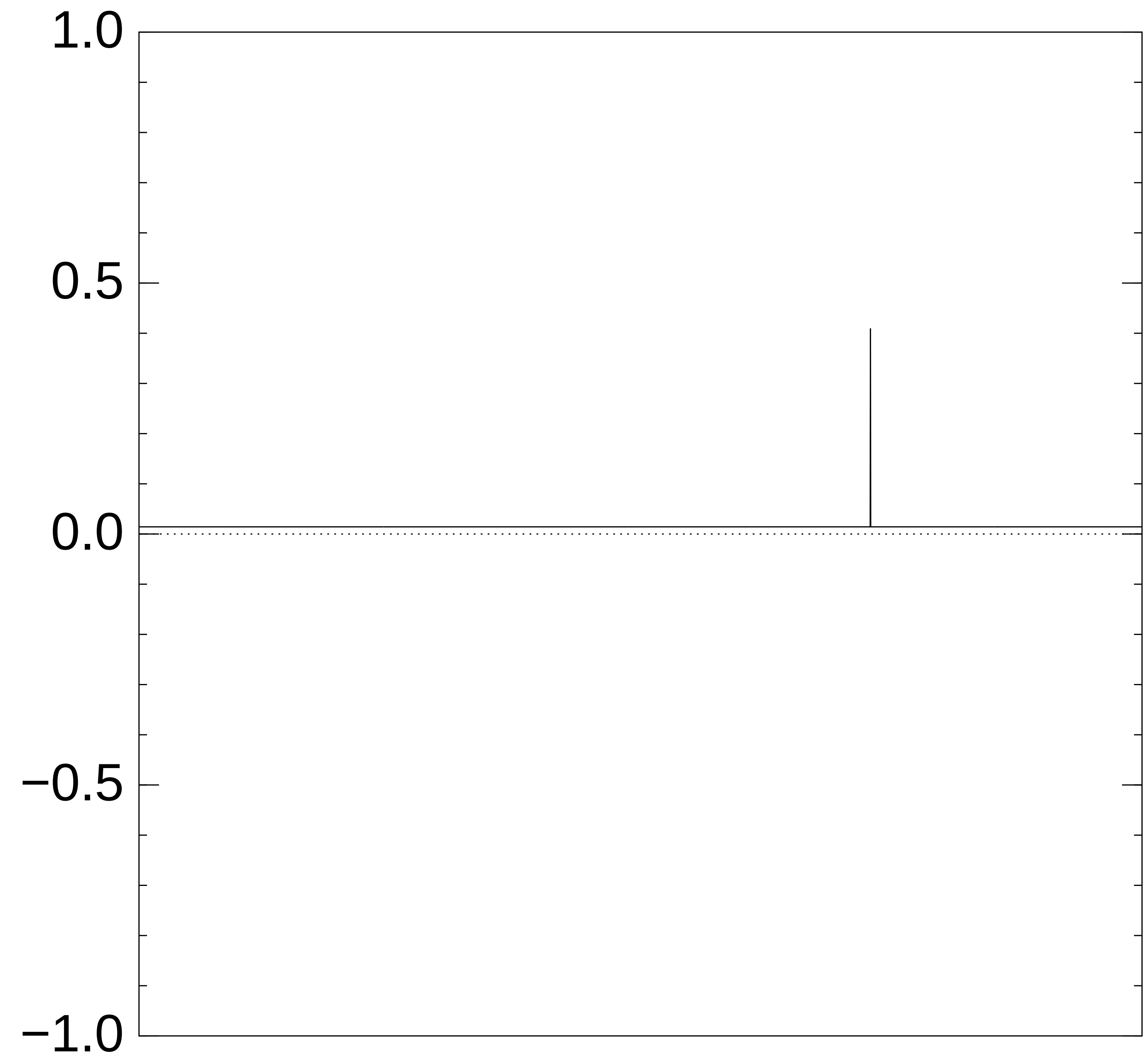
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

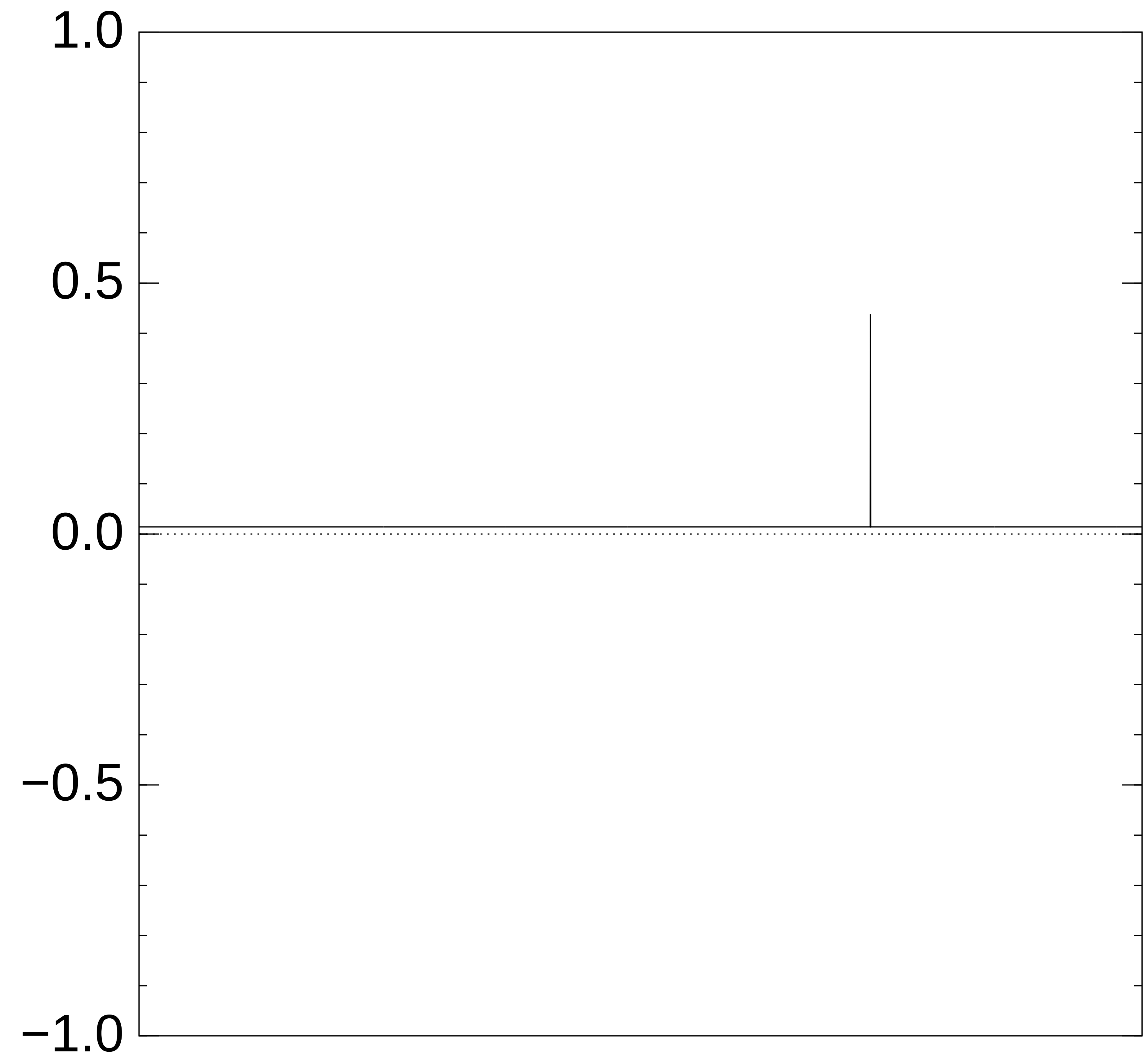
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 $14 \times$ (Step 1 + Step 2):



Start from uniform superposition over n-bit strings u: each $a_u = 1$.

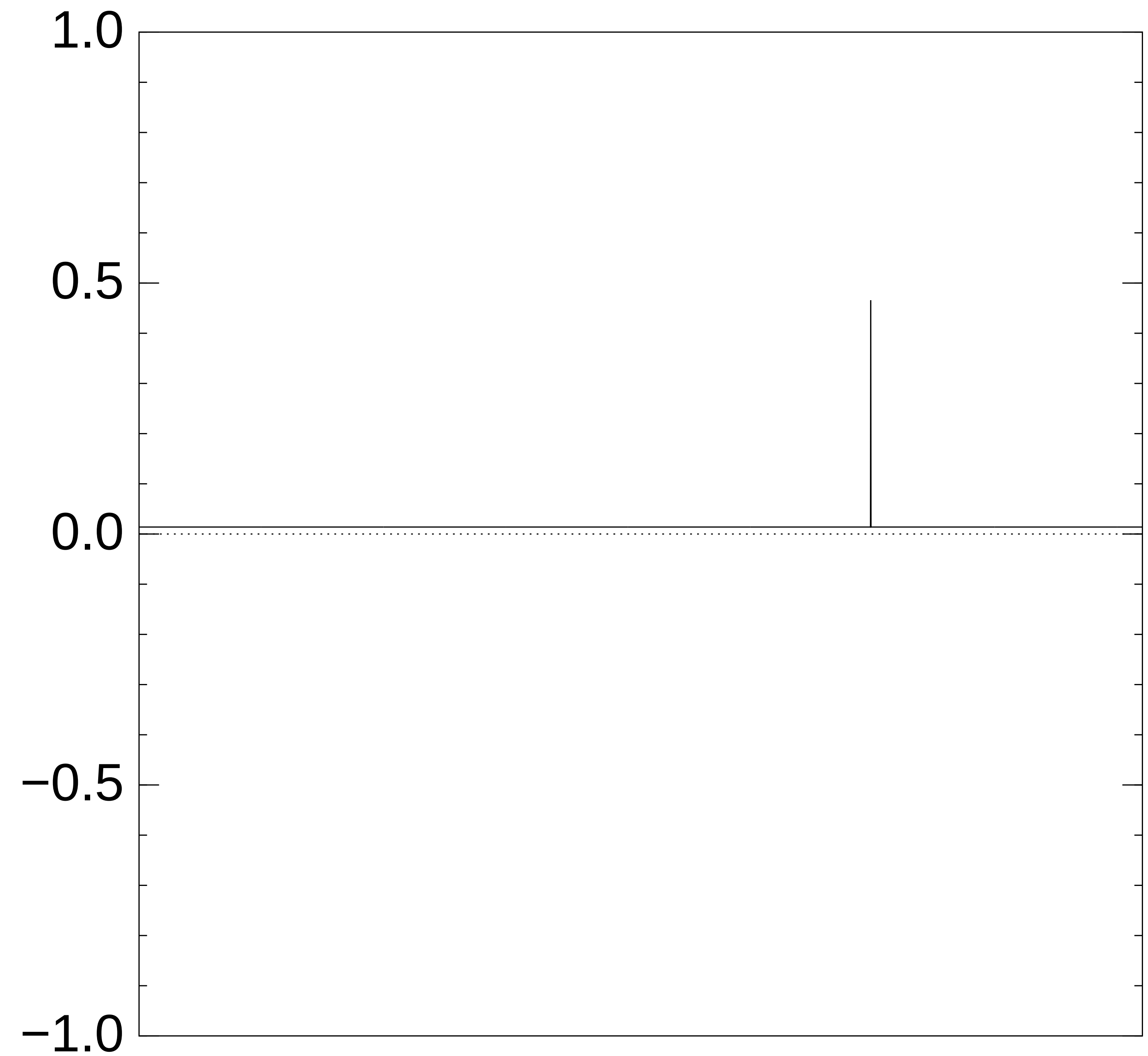
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 $15 \times$ (Step 1 + Step 2):



Start from uniform superposition over n-bit strings u: each $a_u = 1$.

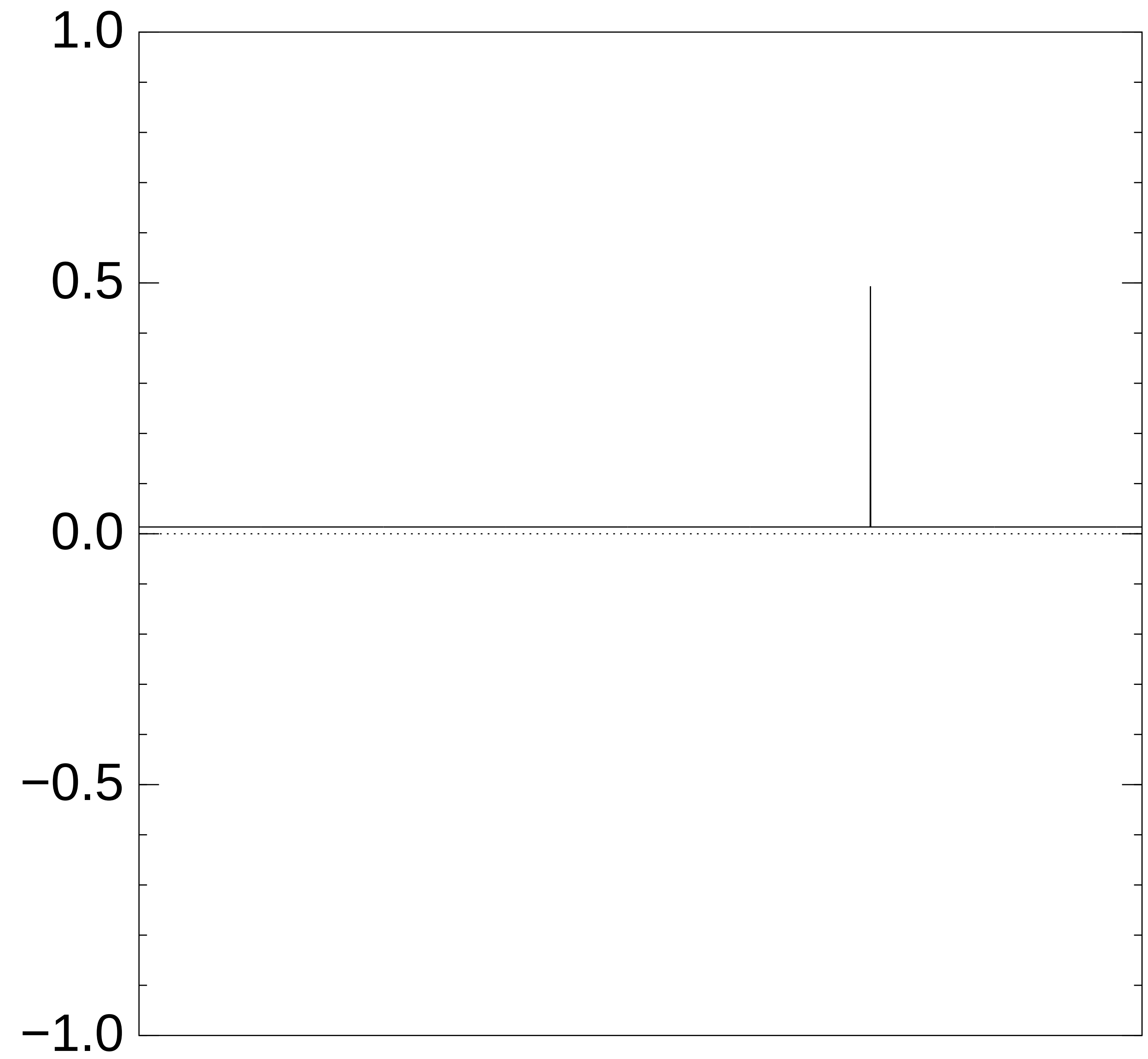
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

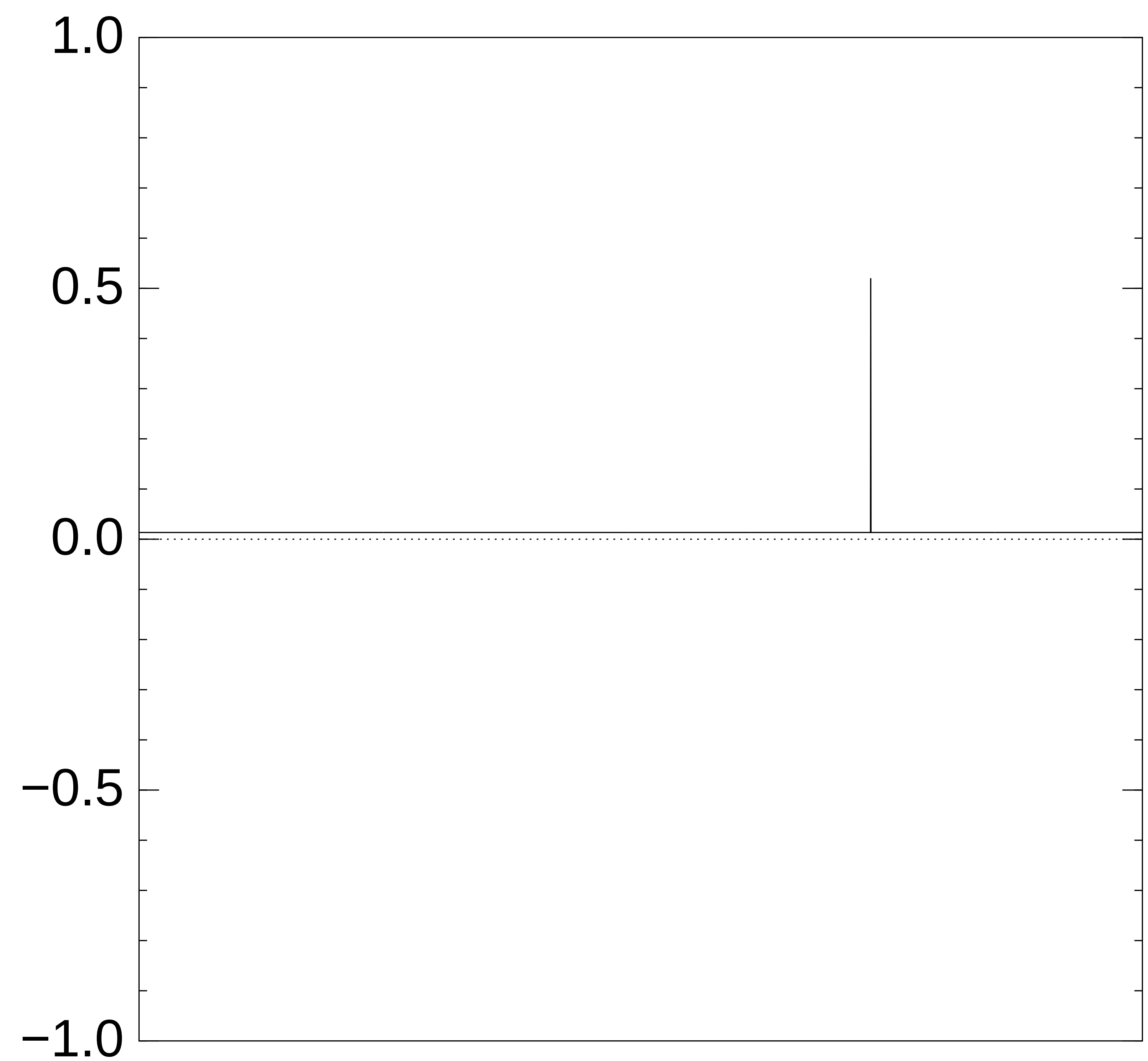
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

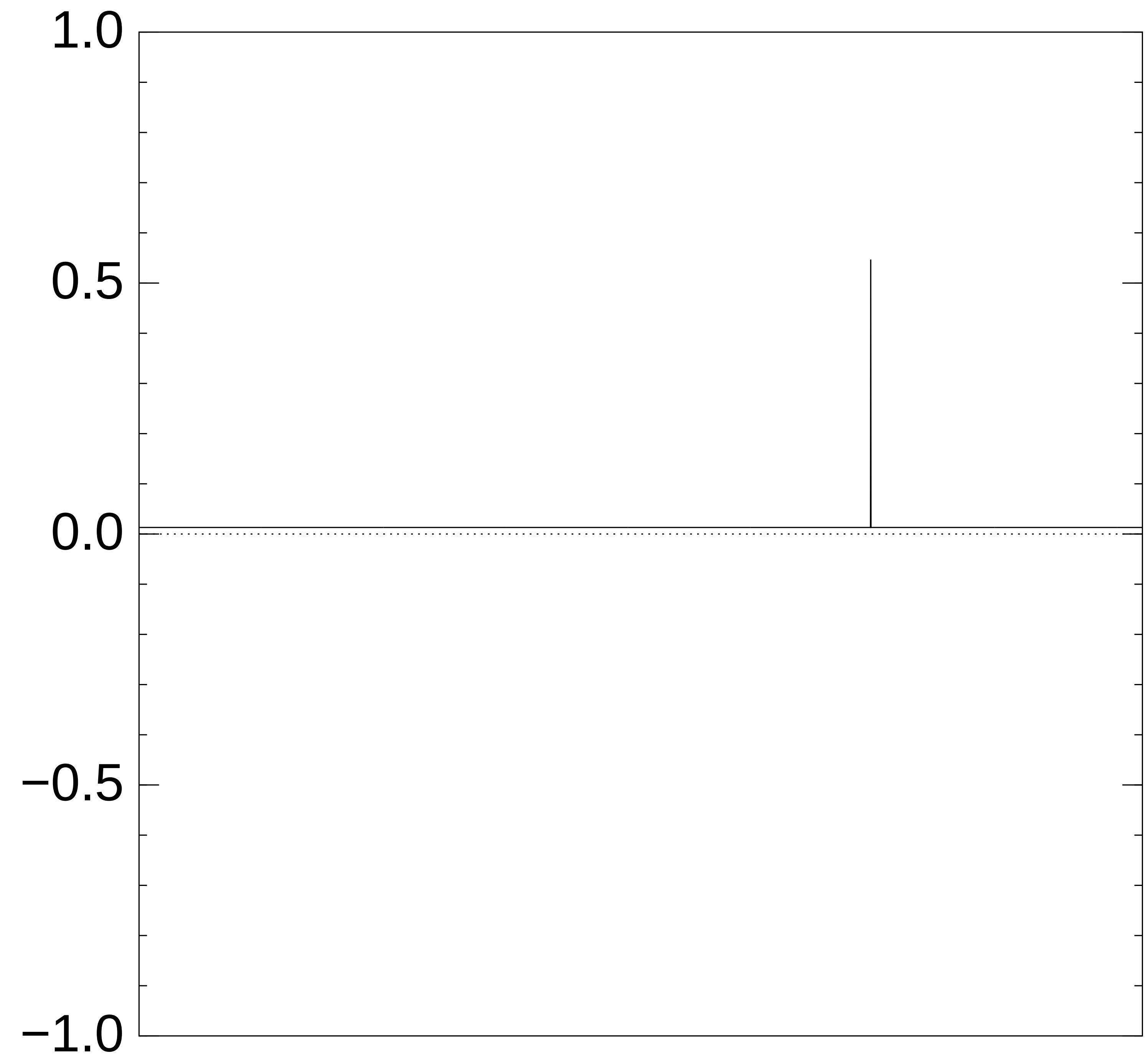
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

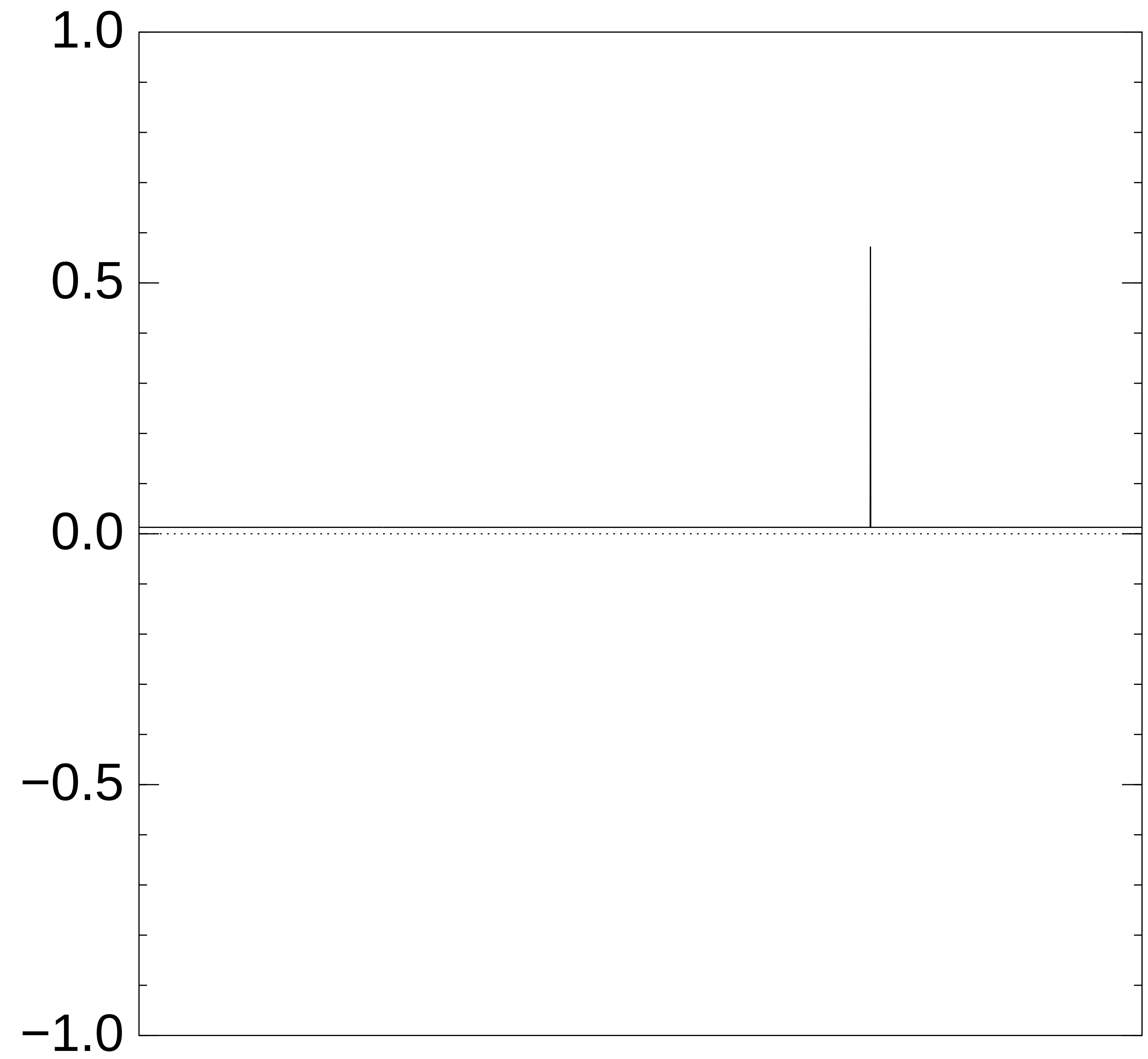
Step 1: Set $a \leftarrow b$ where $b_u = -a_u$ if $f(u) = 0$, $b_u = a_u$ otherwise. This is fast if f is fast.

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 $19 \times (\text{Step } 1 + \text{Step } 2)$:



Start from uniform superposition over n-bit strings u: each $a_u = 1$.

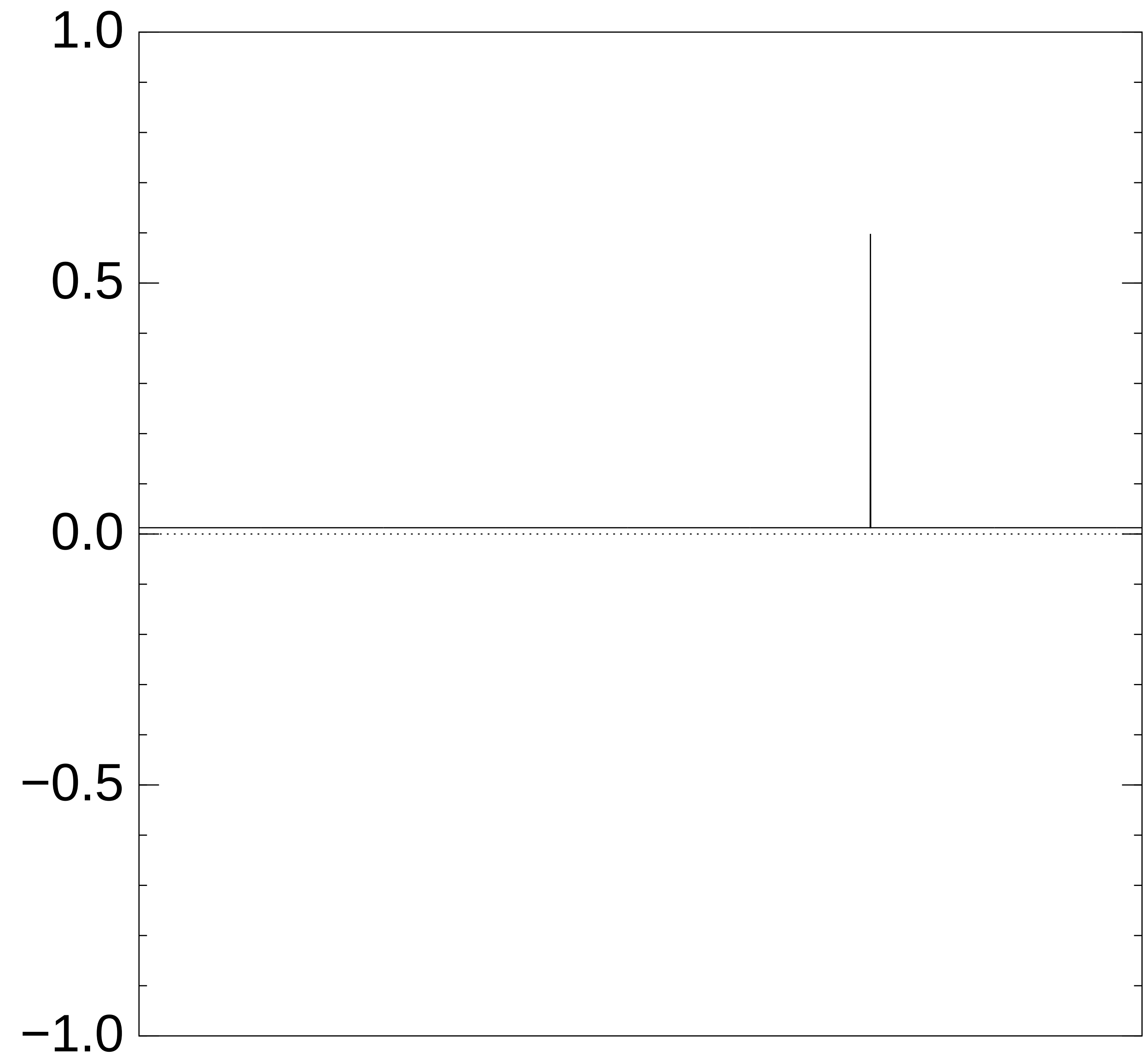
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

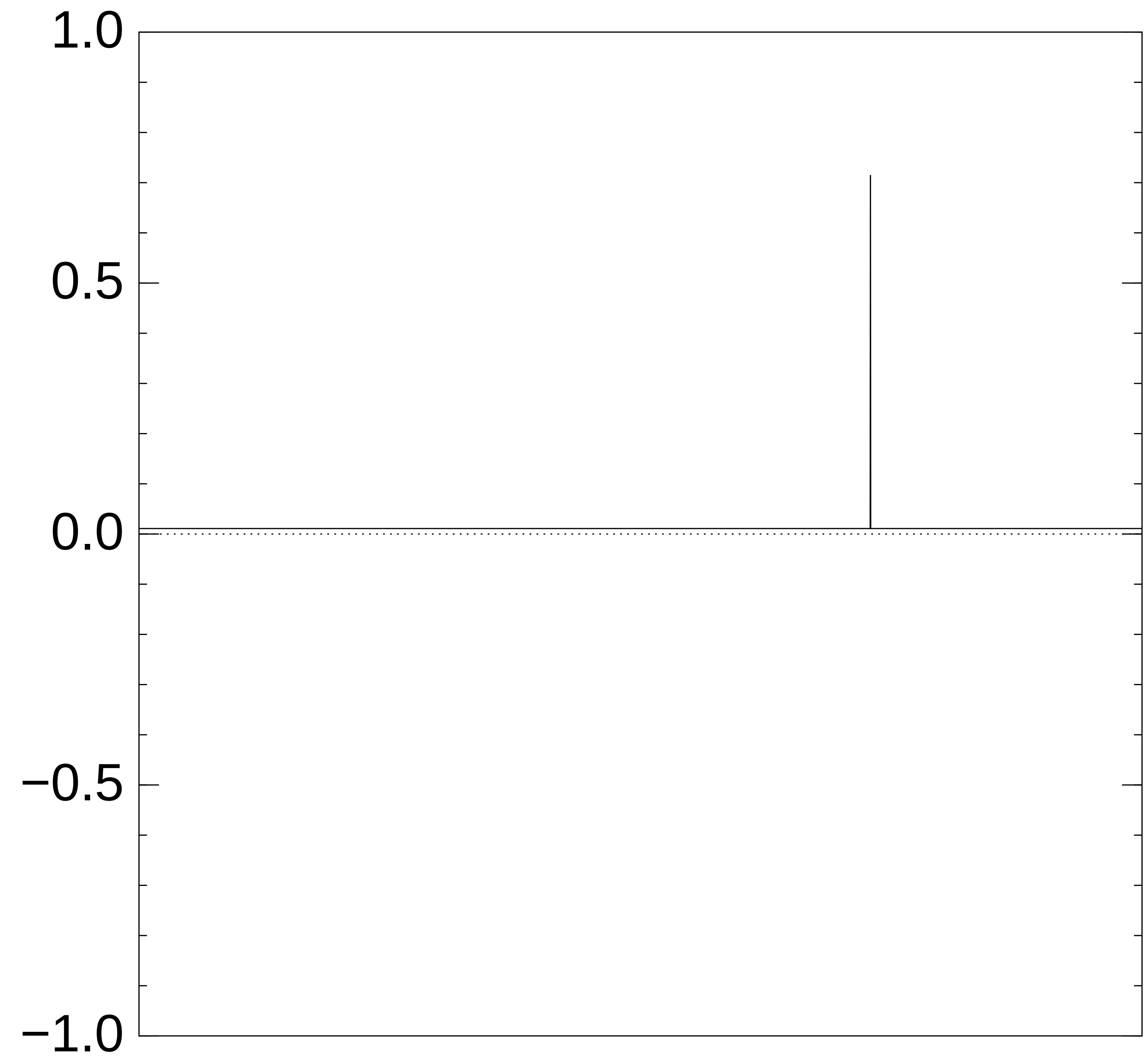
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

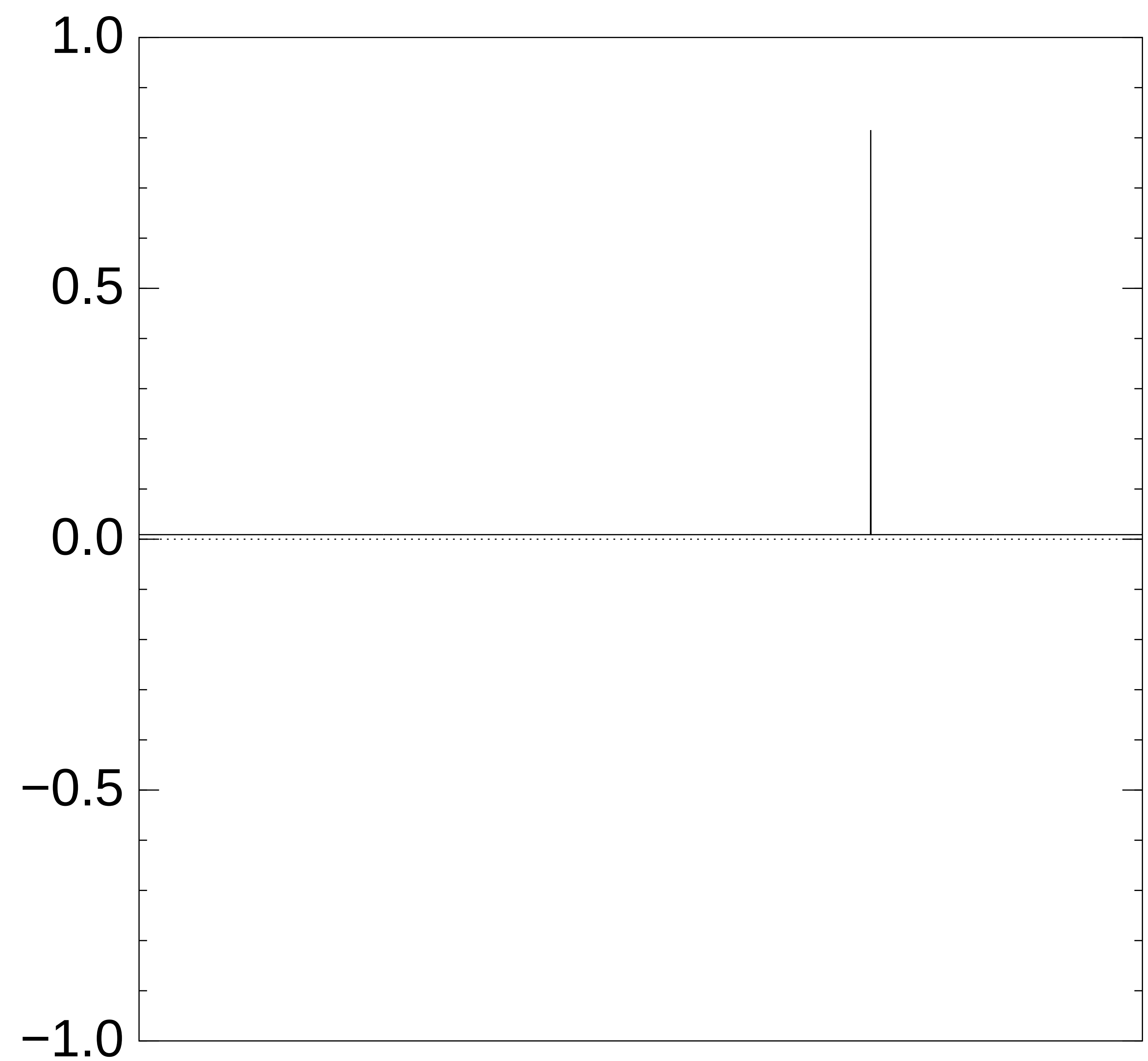
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

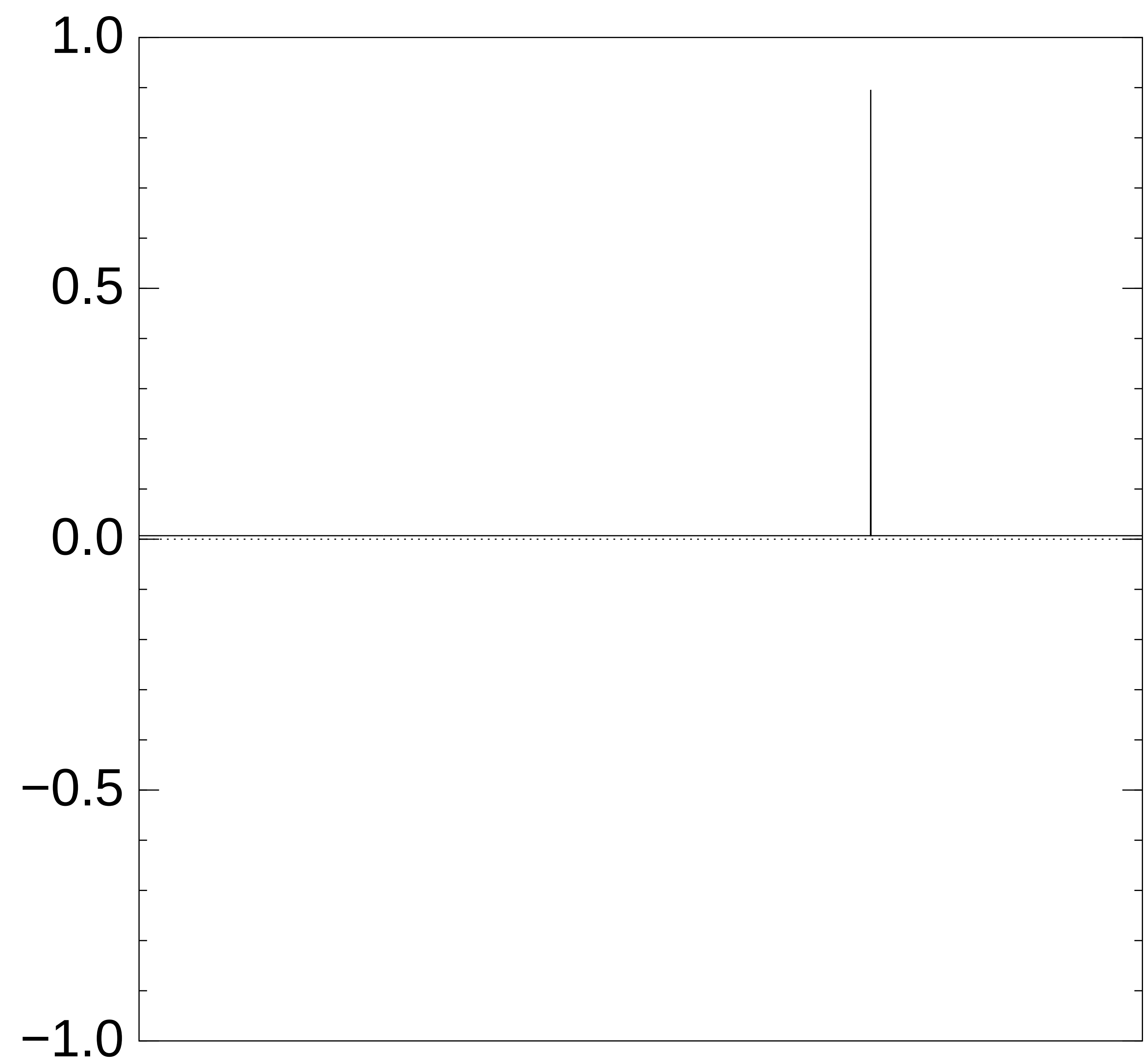
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

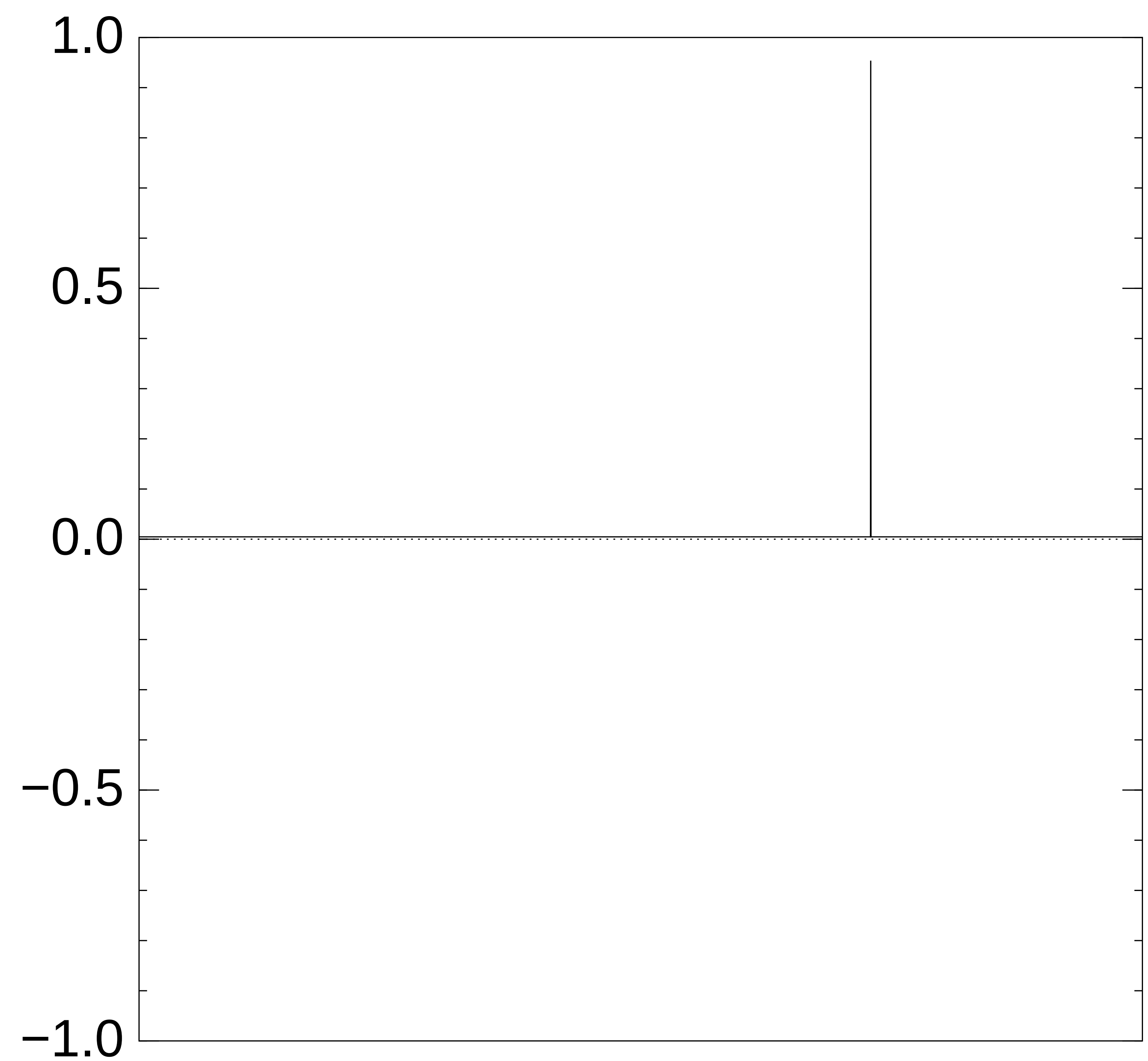
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

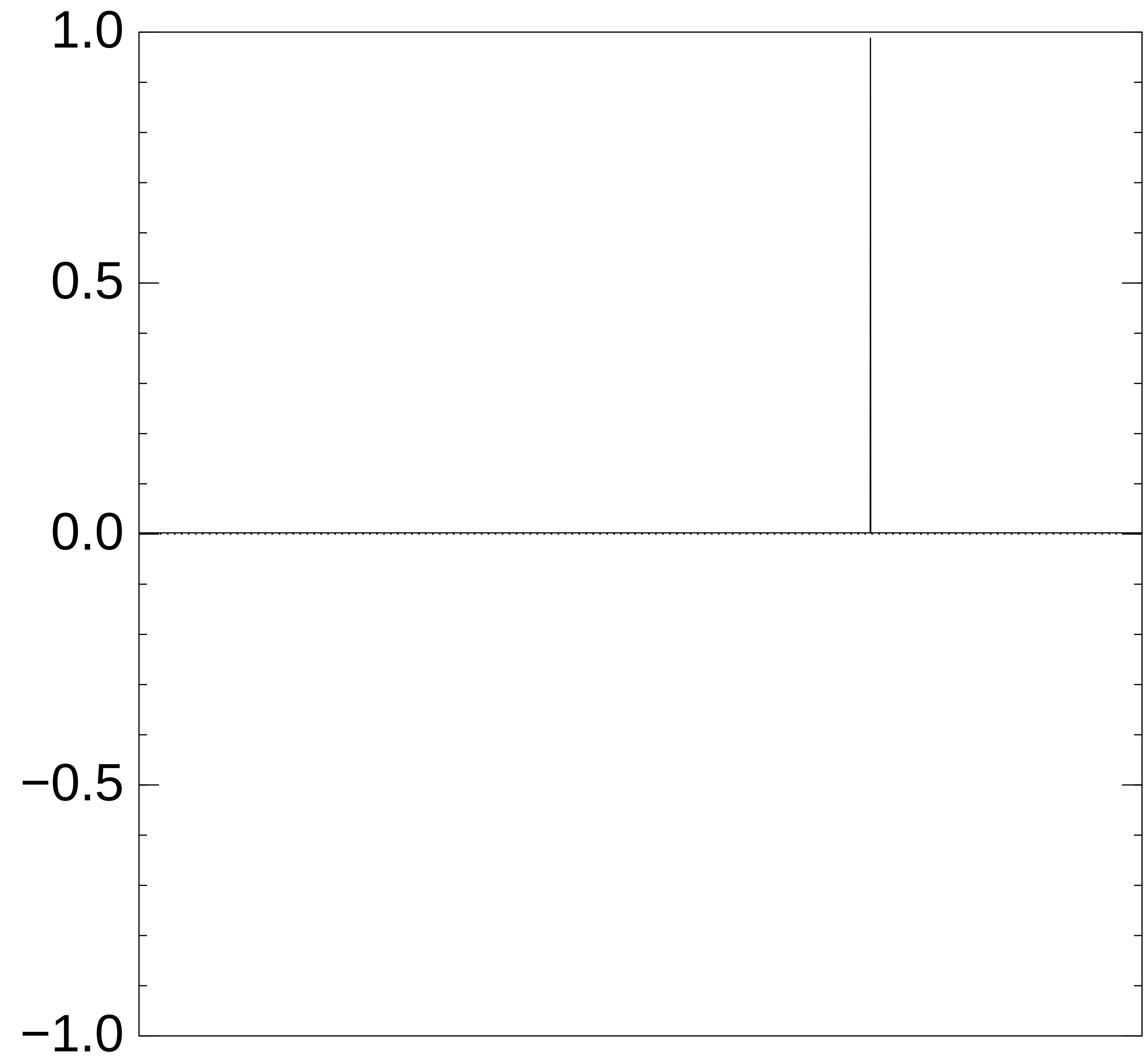
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

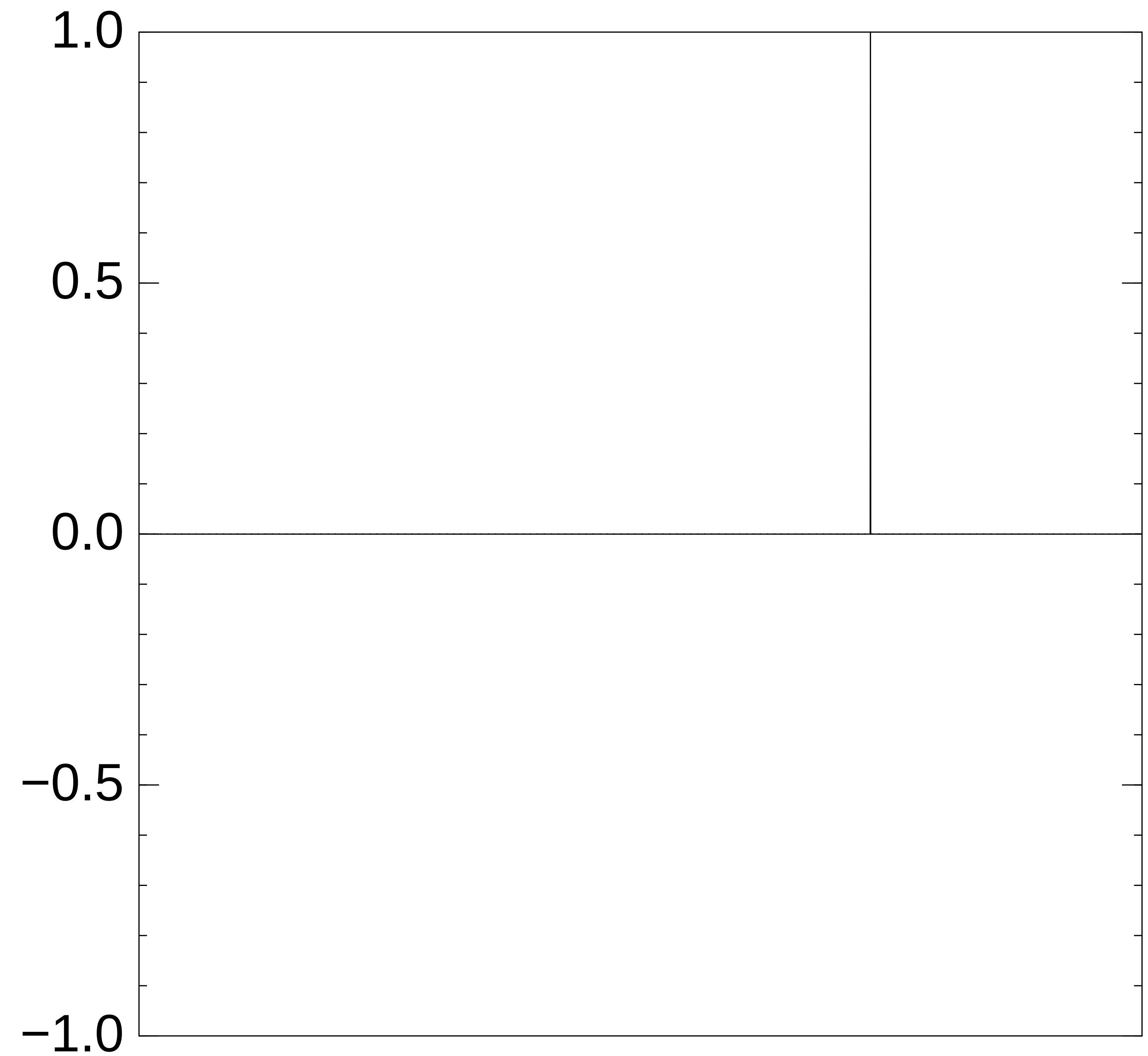
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

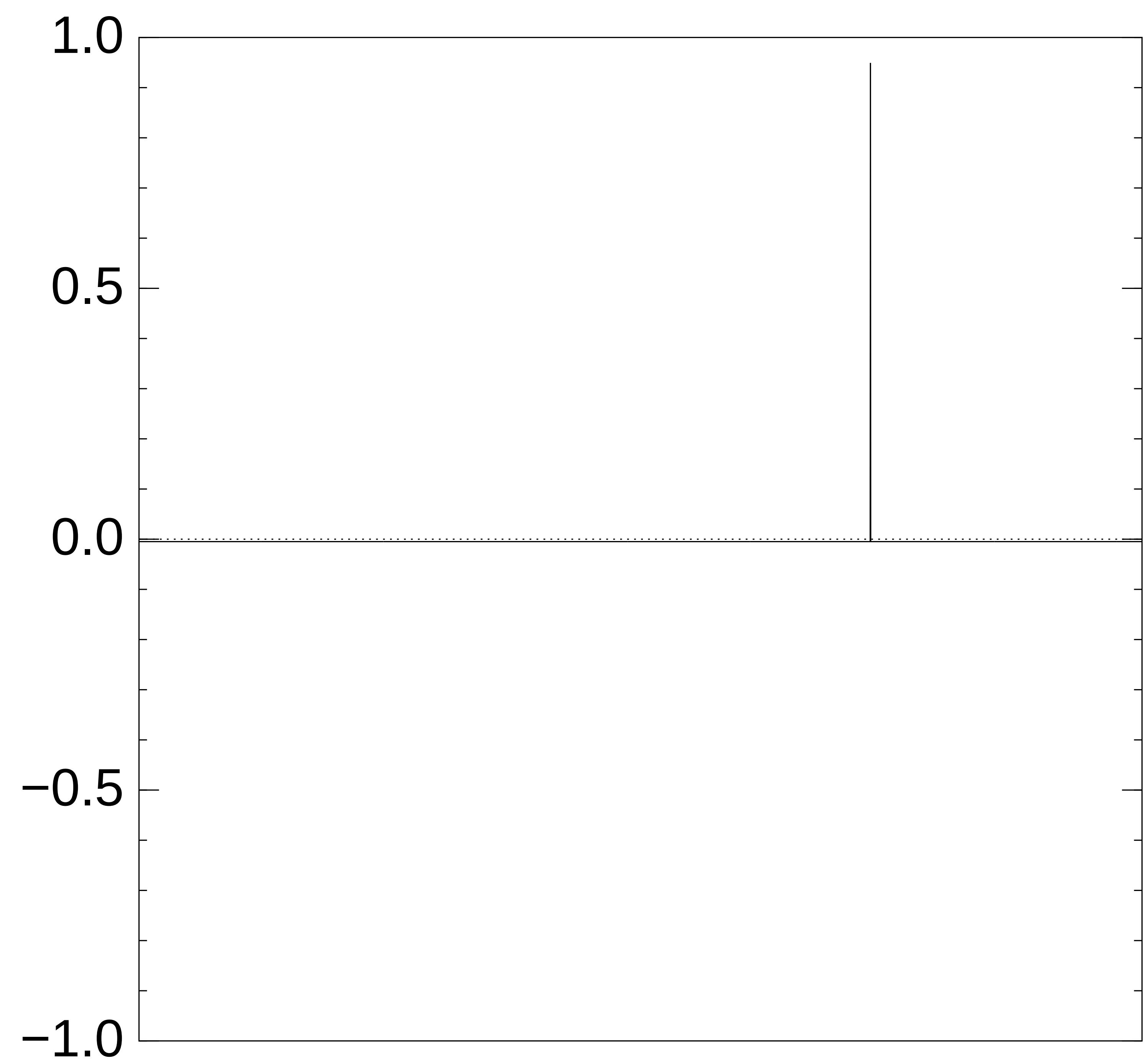
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

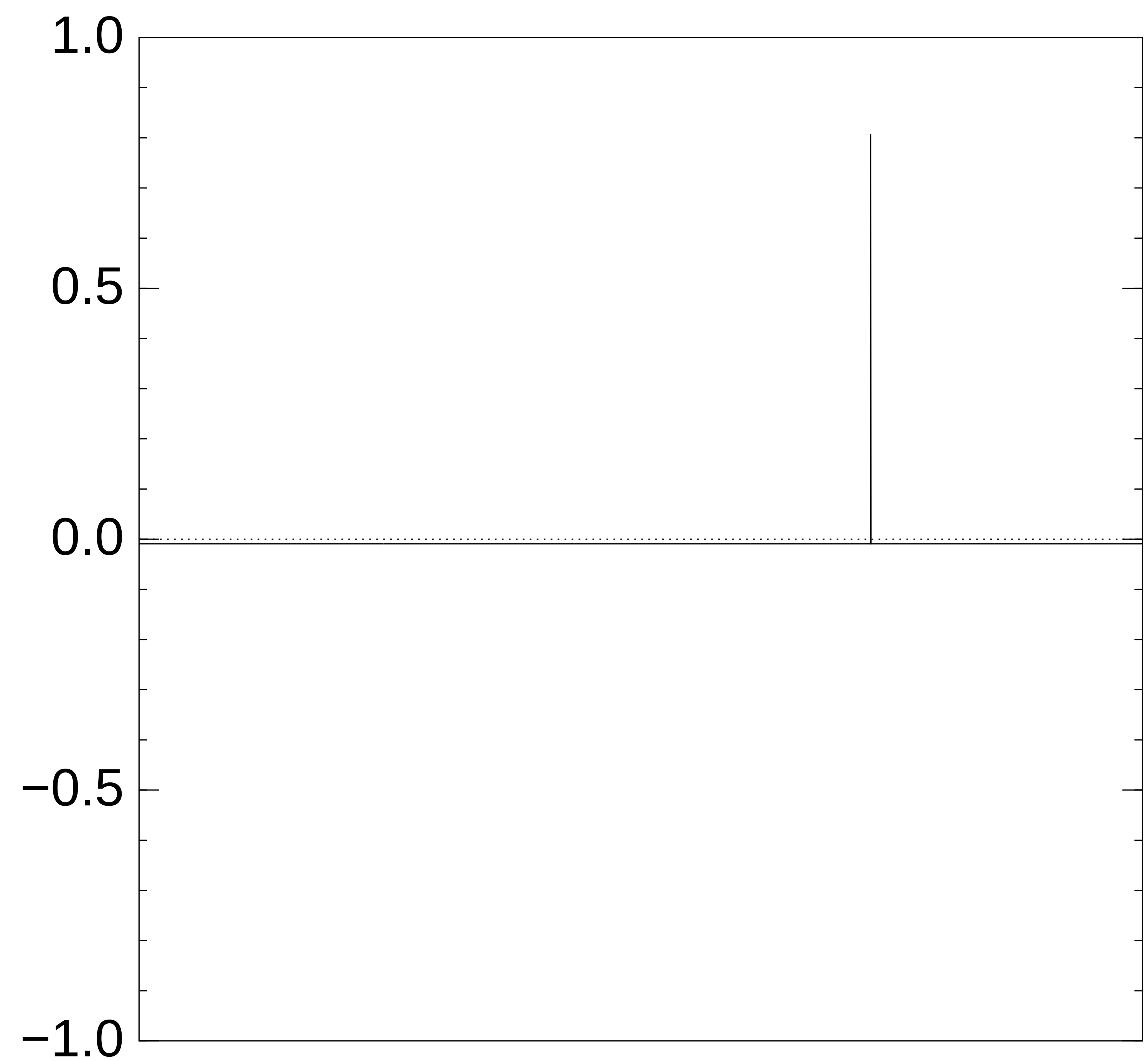
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

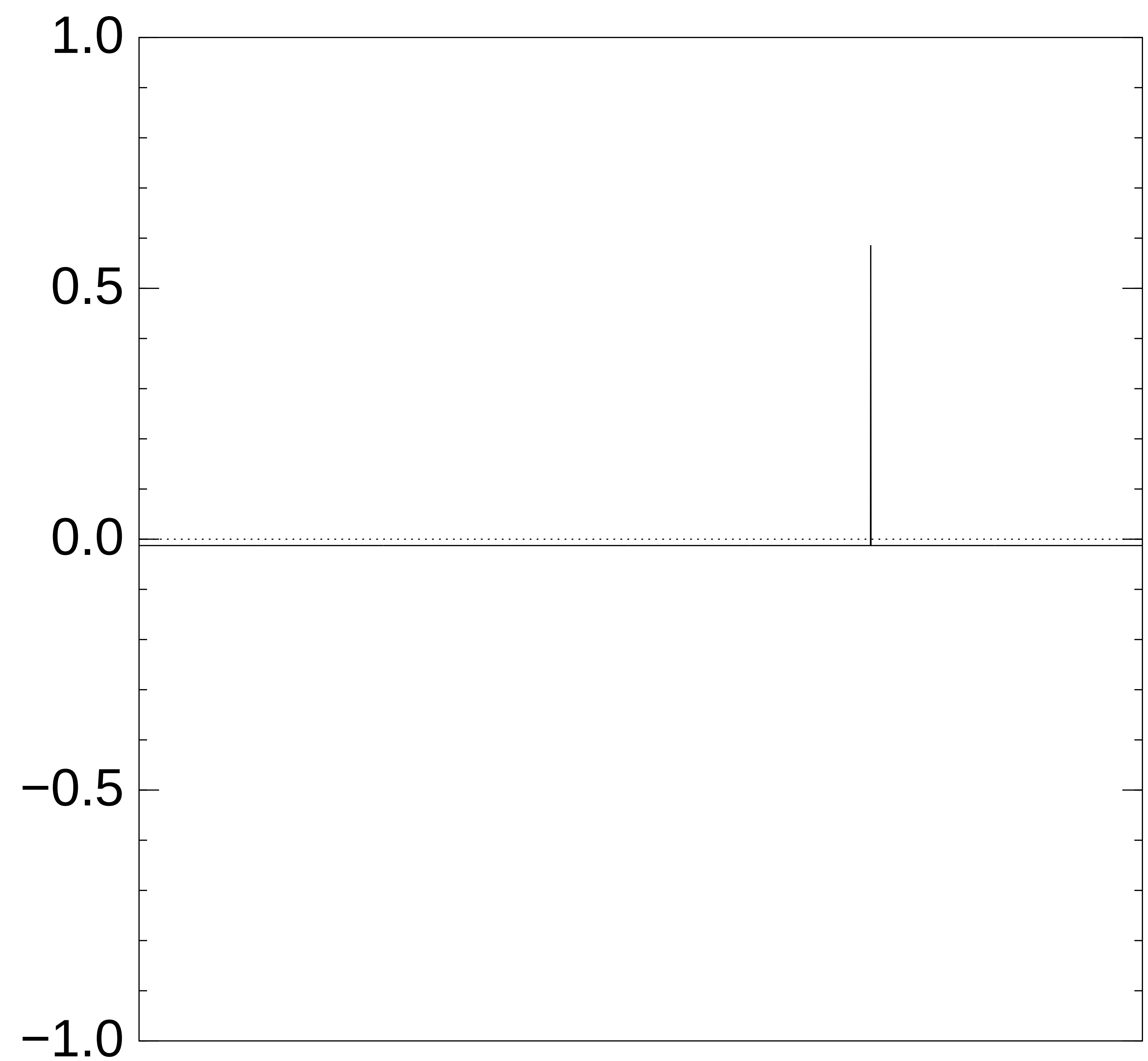
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

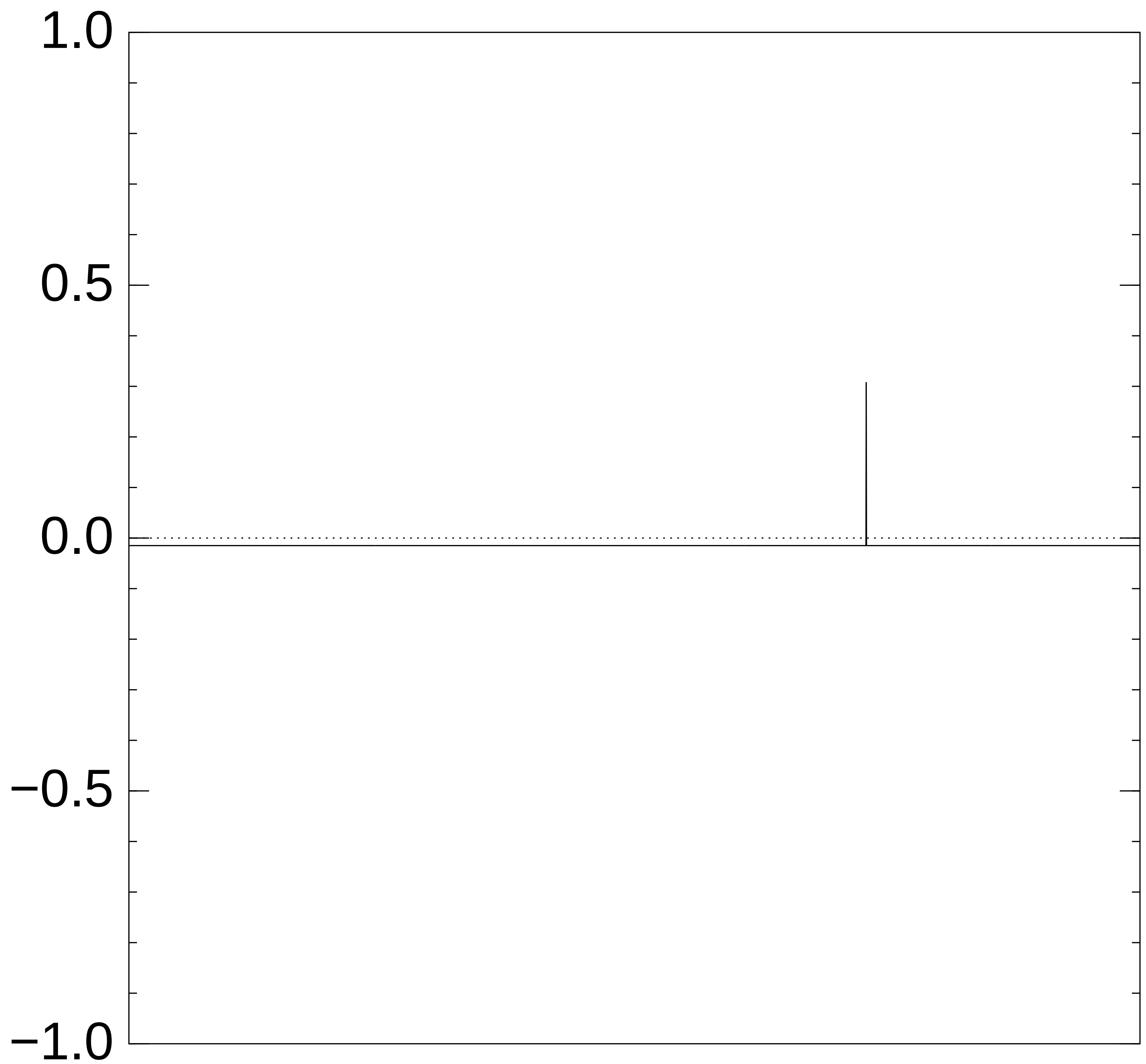
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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 n-bit strings u: each $a_u = 1$.

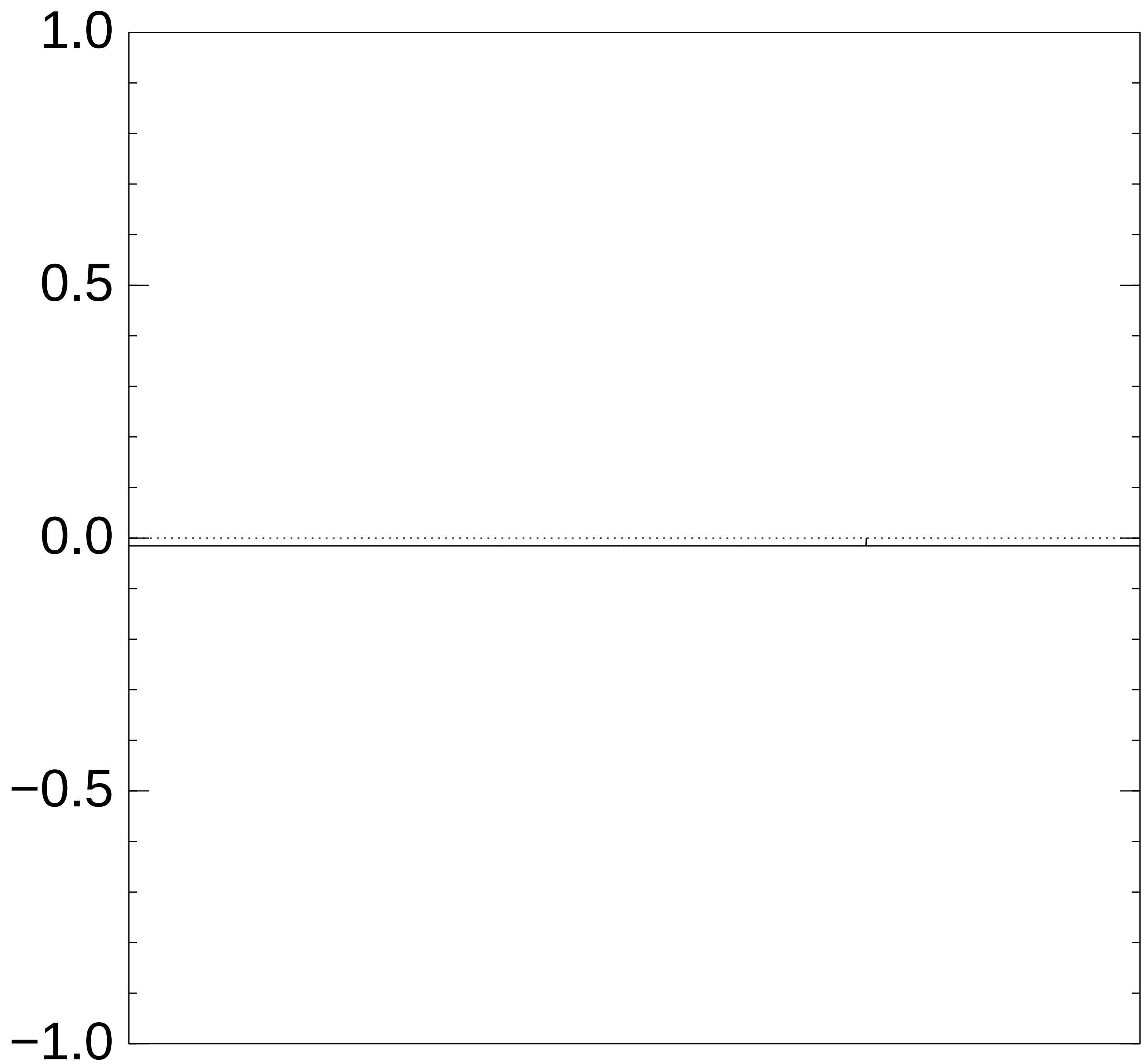
Step 1: Set $a \leftarrow b$ where
$b_u = -a_u$ if $f(u) = 0$,
$b_u = a_u$ otherwise.
This is fast if f is fast.

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
it strings u: each $a_u = 1$.

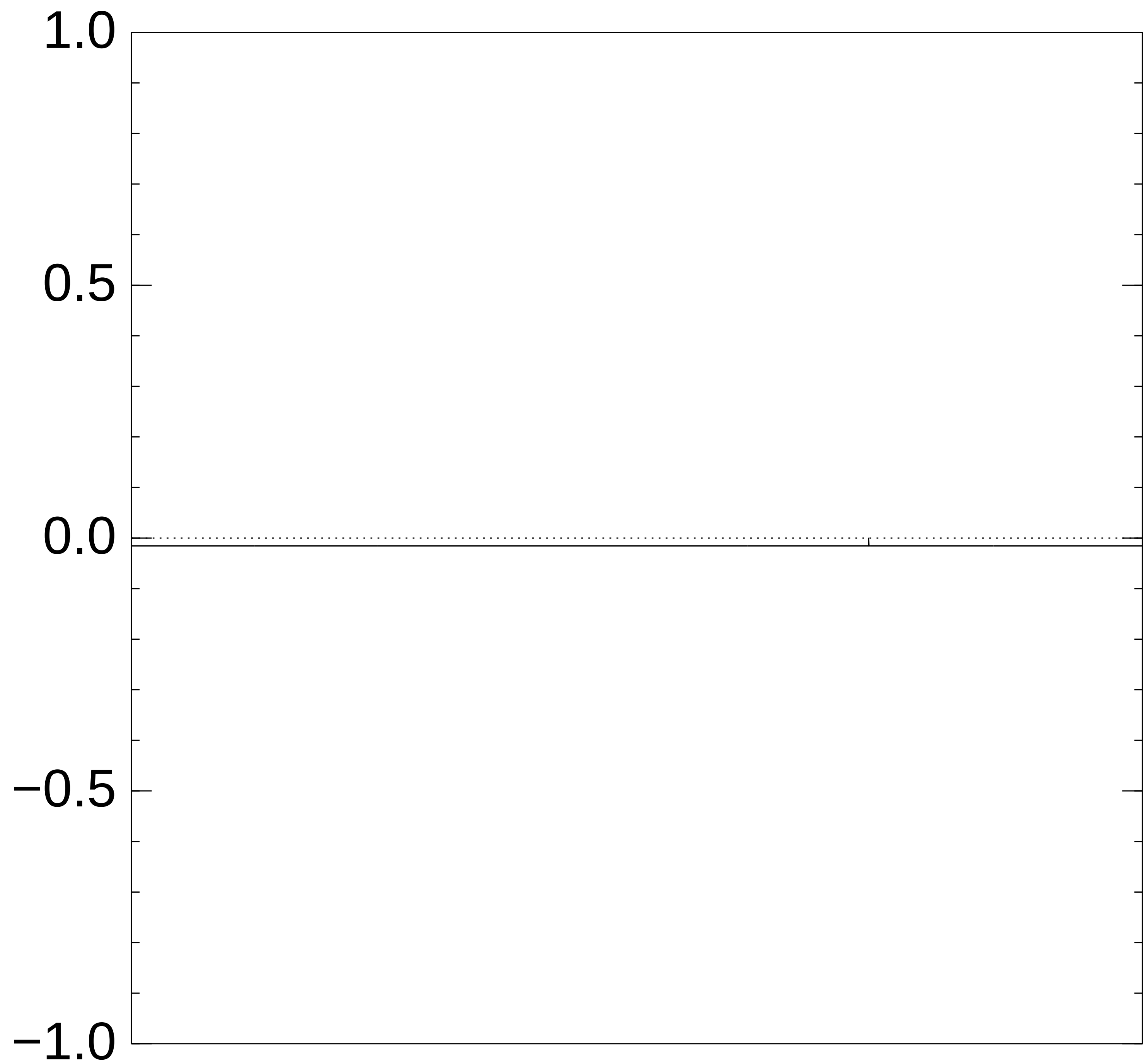
Set $a \leftarrow b$ where
a_u if $f(u) = 0$,
otherwise.
ast if f is fast.

"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
u: each $a_u = 1$.

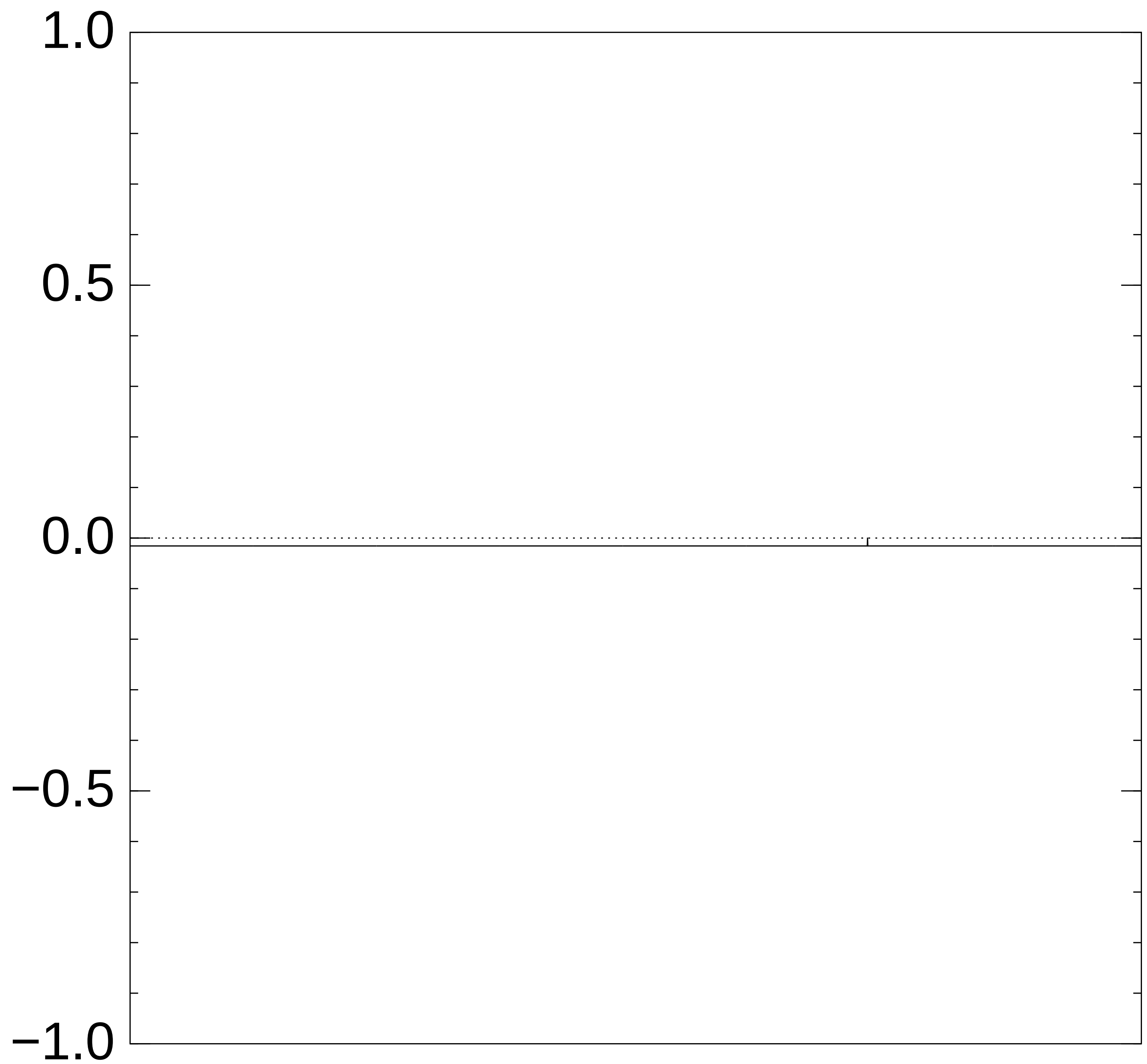
where
$= 0$,
.
fast.

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

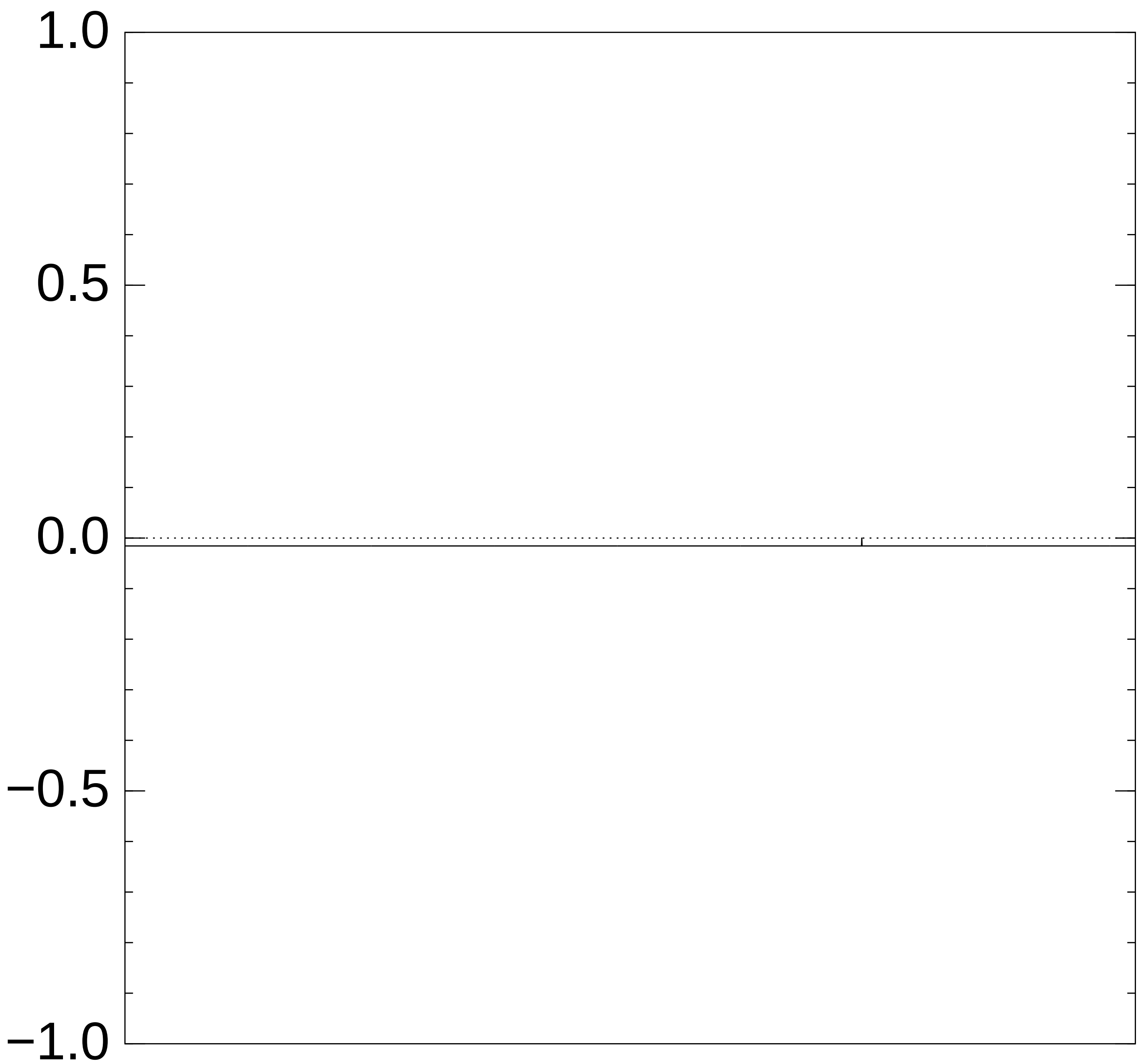
sition

$_u = 1.$

.

nds s.

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



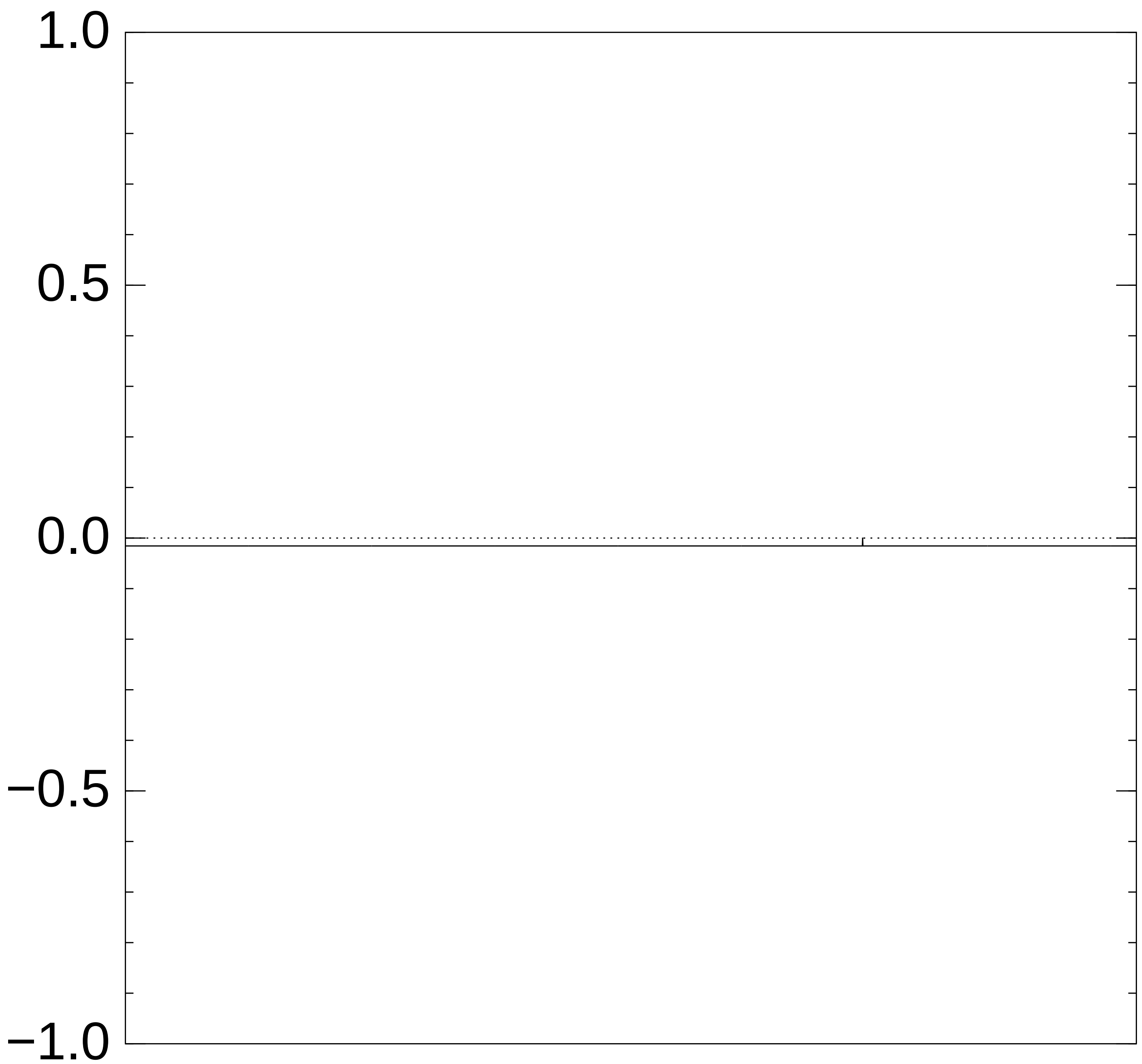
Very bad stopping point.

$u \mapsto a_u$ is completely describ
by a vector of two numbers
(with fixed multiplicities):

(1) a_u for roots u;

(2) a_u for non-roots u.

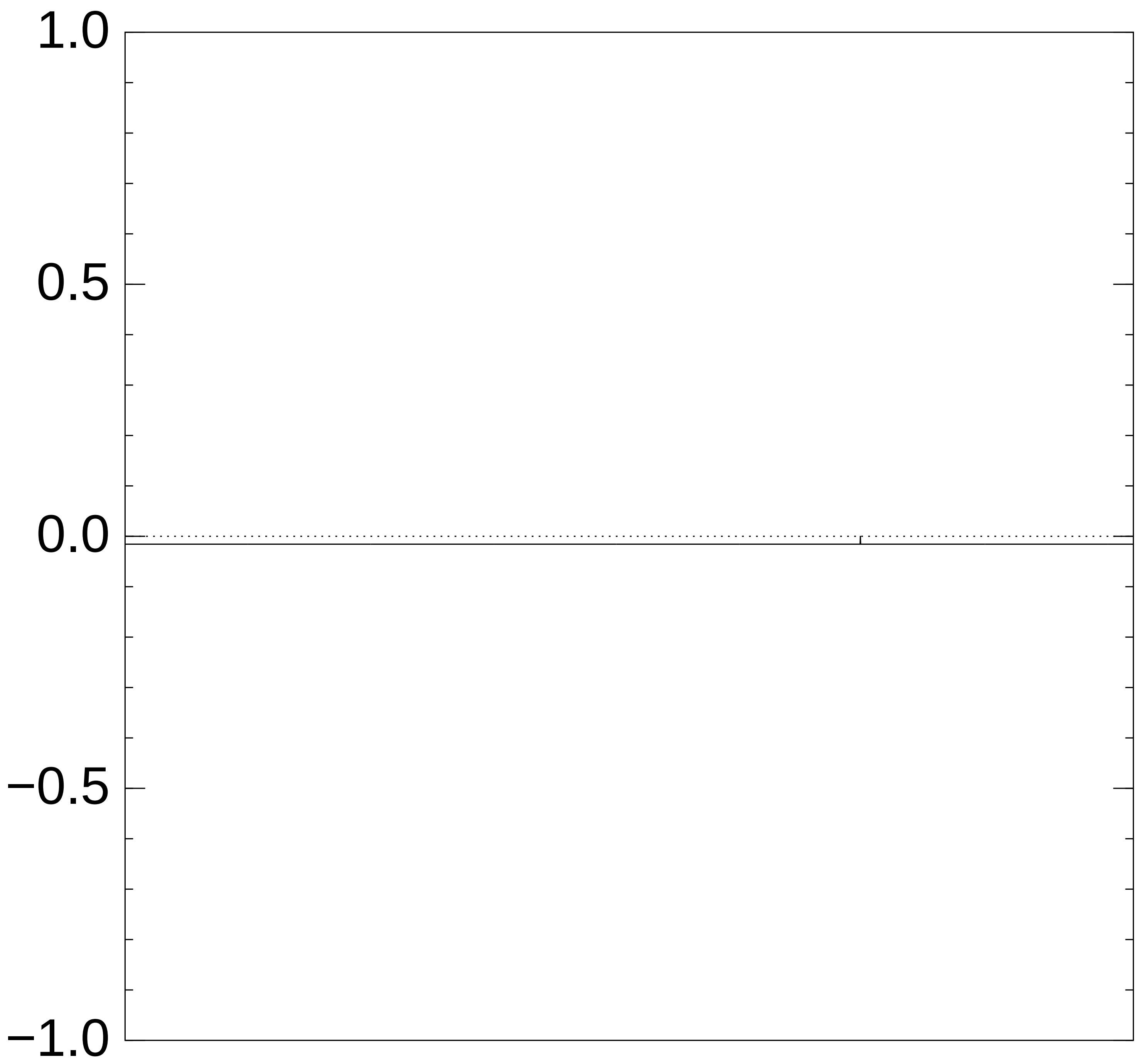
Normalized graph of $u \mapsto a_u$
for an example with $n = 12$
after $100 \times (\text{Step } 1 + \text{Step } 2)$:



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.

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

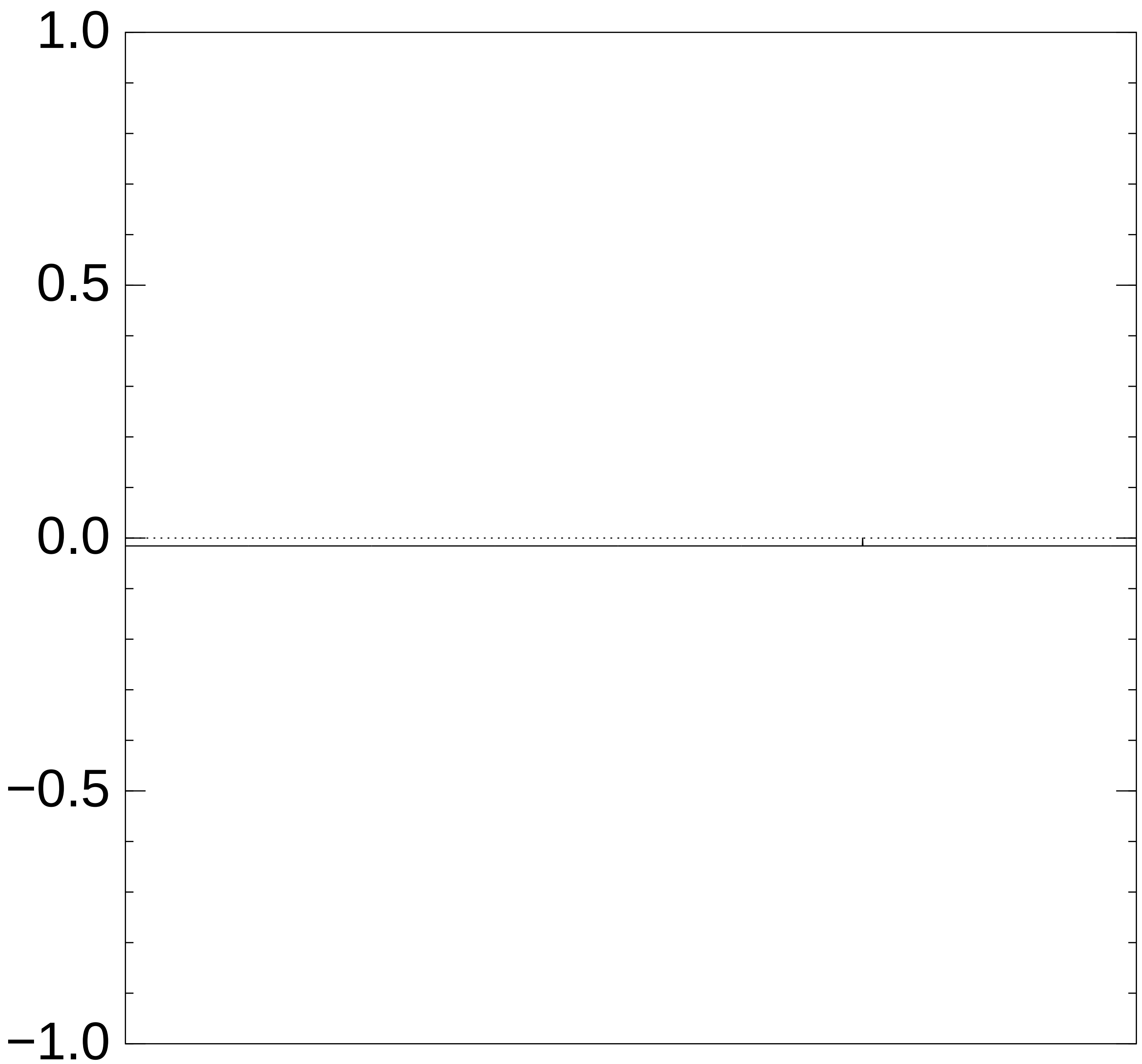


Very bad stopping point.

$u \mapsto a_u$ 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.

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.

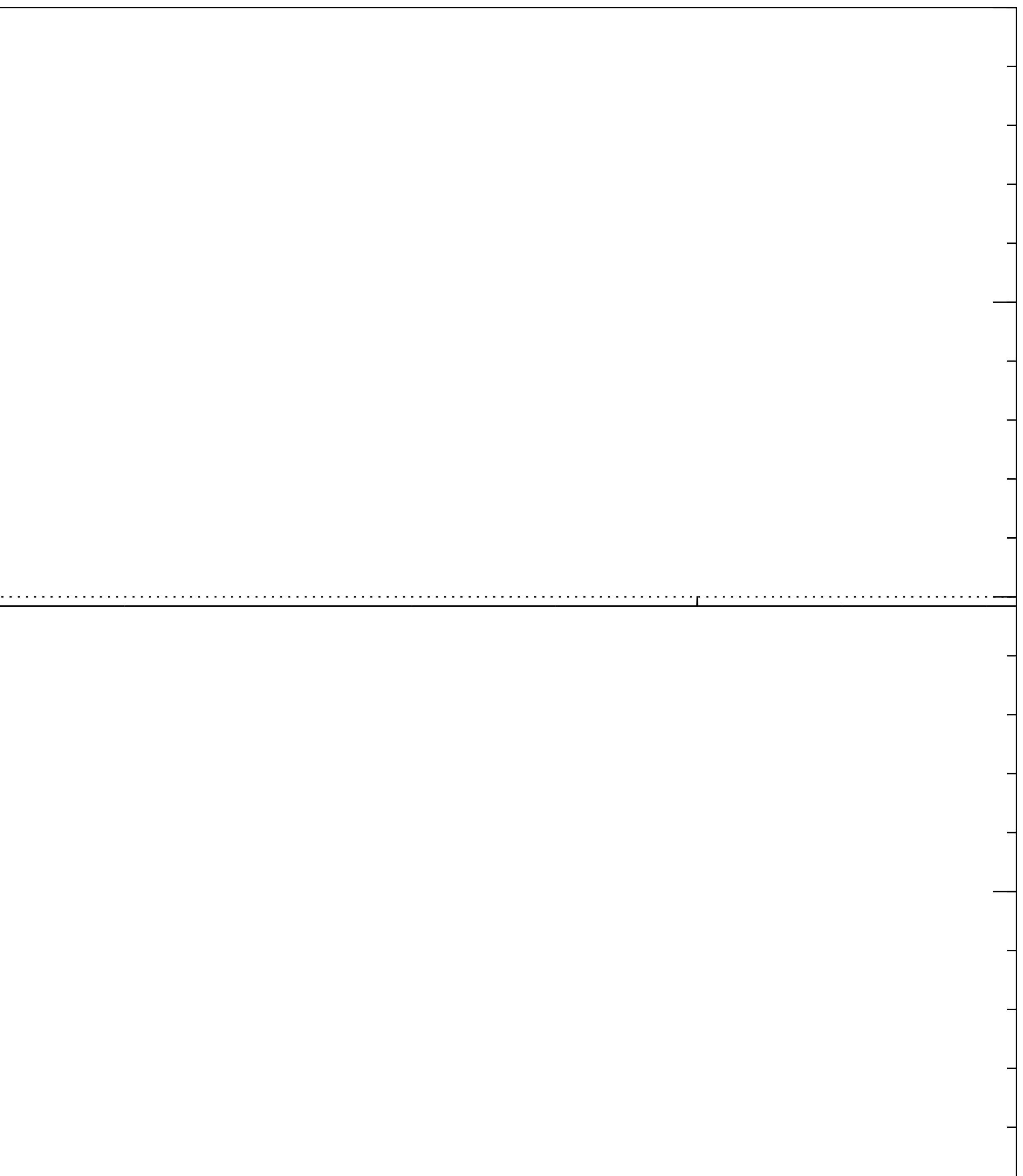
Step 1 + Step 2
act linearly on this vector.

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

Many m

2021: Y

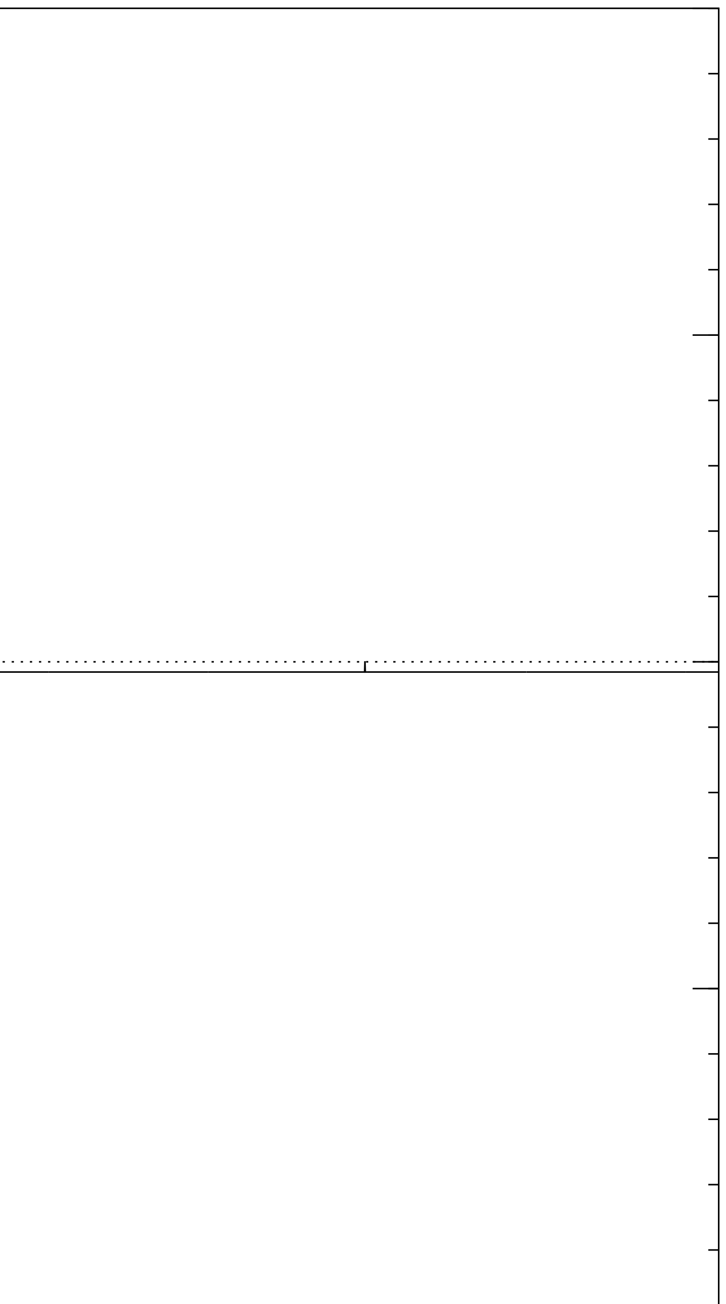
transisto

Can thir

running

as a seq

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.

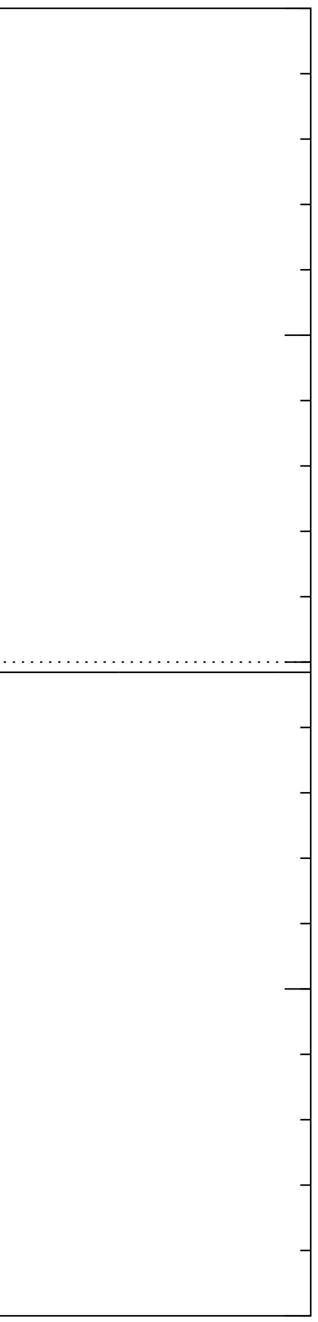
Many more quant

2021: Your CPU c
transistors perform

Can think of any a
running on that C
as a sequence of b

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 quantum algorit

2021: Your CPU consists of
transistors performing bit op

Can think of any algorithm
running on that CPU
as a sequence of bit operatio

$u \mapsto a_u$ 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 quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

$u \mapsto a_u$ 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 quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

$u \mapsto a_u$ 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 quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

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

+ Step 2
rly on this vector.

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

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This sub
algorithn
computa
design n

tely described
numbers
licities):

ts u.

s vector.

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

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includ
algorithms known
computations. Lea
design non-quantu

ped

p

m.

.

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includes the fast
algorithms known for many
computations. Learn how to
design non-quantum algorith

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor “hard” integers, etc. are outside this subset. Learn how to design quantum algorithms!

Many more quantum algorithms

2021: Your CPU consists of transistors performing bit ops.

Can think of any algorithm running on that CPU as a sequence of bit operations.

Can simulate these bit operations and output using NOT, CNOT, CCNOT, and measurement on a quantum computer.

So {non-quantum algorithms} can be viewed as a subset of {quantum algorithms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor "hard" integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

ore quantum algorithms

our CPU consists of
rs performing bit ops.

k of any algorithm
on that CPU
uence of bit operations.

ulate these bit operations
out using NOT, CNOT,
, and measurement
antum computer.

-quantum algorithms}
iewed as a subset of
m algorithms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor “hard” integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Sh

Shor and
techniqu
algorithr
on quant
only two
have bee

um algorithms

consists of
ning bit ops.

lgorithm
PU
it operations.

e bit operations
NOT, CNOT,
surement
nputer.

algorithms}
a subset of
ms}.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor "hard" integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Shor survey

Shor and 1996 Gro
techniques for con
algorithms for clas
on quantum comp
only two significan
have been discove

hms
.
os.
ons.
ations
OT,
s}
f

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor “hard” integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Shor survey regarding
Shor and 1996 Grover: “Th
techniques for constructing
algorithms for classical prob
on quantum computers are
only two significant ones wh
have been discovered so far.

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor "hard" integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

This subset includes the fastest algorithms known for many computations. Learn how to design non-quantum algorithms!

Assuming quantum computers: Fastest known quantum-physics simulators, fastest algorithms to factor “hard” integers, etc. are outside this subset. Learn how to design quantum algorithms!

Often techniques for designing non-quantum algorithms are combined with techniques specific to quantum algorithms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: “These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far.”

2021: Shor’s algorithm and Grover’s algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

oset includes the fastest
ms known for many
ations. Learn how to
on-quantum algorithms!

g quantum computers:
known quantum-physics
rs, fastest algorithms to
hard" integers, etc. are
this subset. Learn how to
uantum algorithms!

echniques for designing
ntum algorithms
bined with techniques
to quantum algorithms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2021: Shor's algorithm and Grover's algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some co

What if

Can try
Analysis
depend

Non-qua
evaluatio
Quantur
quantum

es the fastest
for many
arn how to
um algorithms!

n computers:
antum-physics
algorithms to
gers, etc. are
t. Learn how to
lgorithms!

for designing
rithms
techniques
m algorithms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2021: Shor's algorithm and Grover's algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some common Gr

What if f has mar

Can try same algo
Analysis and optin
depend on $R = \#$

Non-quantum sear
evaluations of f.
Quantum search:
quantum evaluatic

test

o

hms!

ers:

ysics

s to

are

ow to

ng

es

ms.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2021: Shor's algorithm and Grover's algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some common Grover varia

What if f has many roots?

Can try same algorithm. Analysis and optimization depend on $R = \#\{$roots of

Non-quantum search: $\approx 2^n/$ evaluations of f.

Quantum search: $\approx (2^n/R)^1$ quantum evaluations of f.

2001 Shor survey regarding 1994 Shor and 1996 Grover: "These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far."

2021: Shor's algorithm and Grover's algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$ evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$ quantum evaluations of f.

2001 Shor survey regarding 1994 Shor and 1996 Grover: “These techniques for constructing faster algorithms for classical problems on quantum computers are the only two significant ones which have been discovered so far.”

2021: Shor’s algorithm and Grover’s algorithm continue to play critical roles. There are also several useful generalizations and further ideas adding to the landscape of quantum speedups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$ evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$ quantum evaluations of f.

Alternative approach, instead of redoing analysis and optimization: restrict f to a (pseudo)random input set; use unique-root Grover.

or survey regarding 1994
d 1996 Grover: "These
ues for constructing faster
ms for classical problems
tum computers are the
o significant ones which
en discovered so far."

hor's algorithm and
 algorithm continue to
ical roles. There are also
useful generalizations
her ideas adding to the
pe of quantum speedups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if
values o

Can mod
negating
negate w

regarding 1994
over: "These
structing faster
ssical problems
uters are the
nt ones which
red so far."

rithm and
n continue to
. There are also
eralizations
adding to the
tum speedups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if there are
values of f, not ju

Can modify algorit
negating when $f(c$
negate when $g(f($

1994
ese
faster
lems
the
ich
''
to
also
s
the
lups.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if there are many "go
values of f, not just value 0

Can modify algorithm: inste
negating when $f(q) = 0$,
negate when $g(f(q)) = 0$.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if there are many "good"
values of f, not just value 0?

Can modify algorithm: instead of
negating when $f(q) = 0$,
negate when $g(f(q)) = 0$.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization
depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$
evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$
quantum evaluations of f.

Alternative approach, instead of
redoing analysis and optimization:
restrict f to a (pseudo)random
input set; use unique-root Grover.

What if there are many "good"
values of f, not just value 0?

Can modify algorithm: instead of
negating when $f(q) = 0$,
negate when $g(f(q)) = 0$.

Or simply apply original Grover to
the composition $q \mapsto g(f(q))$.

Some common Grover variants

What if f has many roots?

Can try same algorithm.
Analysis and optimization depend on $R = \#\{\text{roots of } f\}$.

Non-quantum search: $\approx 2^n/R$ evaluations of f.
Quantum search: $\approx (2^n/R)^{1/2}$ quantum evaluations of f.

Alternative approach, instead of redoing analysis and optimization: restrict f to a (pseudo)random input set; use unique-root Grover.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

ommon Grover variants

f has many roots?

same algorithm.
and optimization
on $R = \#\{\text{roots of } f\}$.

ntum search: $\approx 2^n/R$
ons of f.
m search: $\approx (2^n/R)^{1/2}$
n evaluations of f.

ive approach, instead of
analysis and optimization:
f to a (pseudo)random
t; use unique-root Grover.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

More int
quantum
powerful

Say $u \mapsto$
but have
$u, u', f($
specified
Want to

Non-qua
Start wi
Replace
repeat e
check if

Quantur

over variants

ny roots?

rithm.
nization
$\{$roots of $f\}$.

rch: $\approx 2^n/R$

$\approx (2^n/R)^{1/2}$
ons of f.

nch, instead of
nd optimization:
eudo)random
que-root Grover.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

More interesting g
quantum walks. S
powerful than orig

Say $u \mapsto f(u)$ isn't
but have a very fa
$u, u', f(u) \mapsto f(u'$
specified set of "n
Want to find "goo

Non-quantum ranc
Start with one u;
Replace u by rand
repeat enough tim
check if good; kee

Quantum walk: (r

nts

$f\}$.

R

$/2$

d of

zation:

om

rover.

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

More interesting generalizati
quantum walks. Seems more
powerful than original Grove

Say $u \mapsto f(u)$ isn't very fast
but have a very fast algorith
$u, u', f(u) \mapsto f(u')$ for u' in
specified set of "neighbors"
Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute f
Replace u by random neighb
repeat enough times for mix
check if good; keep repeatin

Quantum walk: (repetitions

What if there are many "good" values of f, not just value 0?

Can modify algorithm: instead of negating when $f(q) = 0$, negate when $g(f(q)) = 0$.

Or simply apply original Grover to the composition $q \mapsto g(f(q))$.

What if one doesn't know R?

Can modify algorithm. Or repeat original algorithm with sequence of guesses for R, starting with 2^n and decreasing exponentially. Approximation of R suffices.

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute $f(u)$.
Replace u by random neighbor; repeat enough times for mixing; check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

there are many "good"
f f, not just value 0?

dify algorithm: instead of
g when $f(q) = 0$,
when $g(f(q)) = 0$.

ly apply original Grover to
position $q \mapsto g(f(q))$.

one doesn't know R?

dify algorithm. Or repeat
algorithm with sequence
es for R, starting with
decreasing exponentially.
mation of R suffices.

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute $f(u)$.
Replace u by random neighbor;
repeat enough times for mixing;
check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

Extreme
Complet
Recomp
Mixes in

many "good"
st value 0?

thm: instead of
$q) = 0$,
$q)) = 0$.

iginal Grover to
$\mapsto g(f(q))$.

't know R?

thm. Or repeat
with sequence
starting with
exponentially.
R suffices.

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk: Start with one u; compute $f(u)$. Replace u by random neighbor; repeat enough times for mixing; check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

Extreme example
Completely unrest
Recompute f at e
Mixes instantly. S

od"
?
ead of

ver to
)).
?

epeat
ence
th
ially.
.

More interesting generalization:
quantum walks. Seems more
powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast
but have a very fast algorithm
$u, u', f(u) \mapsto f(u')$ for u' in a
specified set of "neighbors" of u.
Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute $f(u)$.
Replace u by random neighbor;
repeat enough times for mixing;
check if good; keep repeating.

Quantum walk: (repetitions)$^{1/2}$.

Extreme example of walk:
Completely unrestricted neig
Recompute f at each step.
Mixes instantly. Same as Gr

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute $f(u)$.
Replace u by random neighbor;
repeat enough times for mixing;
check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting generalization: quantum walks. Seems more powerful than original Grover.

Say $u \mapsto f(u)$ isn't very fast but have a very fast algorithm $u, u', f(u) \mapsto f(u')$ for u' in a specified set of "neighbors" of u. Want to find "good" $f(u)$.

Non-quantum random walk:
Start with one u; compute $f(u)$.
Replace u by random neighbor; repeat enough times for mixing; check if good; keep repeating.

Quantum walk: $(\text{repetitions})^{1/2}$.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs, exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm: nearly 2^n calls to f.
Ambainis, using quantum walk: $\approx 2^{2n/3}$ calls to f.

teresting generalization:
n walks. Seems more
l than original Grover.

→ $f(u)$ isn't very fast
e a very fast algorithm
$u) \mapsto f(u')$ for u' in a
l set of "neighbors" of u.
o find "good" $f(u)$.

ntum random walk:
th one u; compute $f(u)$.
u by random neighbor;
nough times for mixing;
good; keep repeating.

m walk: (repetitions)$^{1/2}$.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs,
exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm:
nearly 2^n calls to f.
Ambainis, using quantum walk:
$\approx 2^{2n/3}$ calls to f.

Sketch c

For $S \subseteq$
define φ
$\tau = \#\{$
T is the

Define "
Chance

To walk
delete o

Non-qua
then inn
Quantur
Take σ t

generalization:
eems more
inal Grover.

t very fast
st algorithm
) for u' in a
eighbors" of u.
od" $f(u)$.

dom walk:
compute $f(u)$.
om neighbor;
es for mixing;
p repeating.

epetitions)$^{1/2}$.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs,
exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm:
nearly 2^n calls to f.
Ambainis, using quantum walk:
$\approx 2^{2n/3}$ calls to f.

Sketch of Ambaini

For $S \subseteq \{\text{inputs}\}$
define $\varphi(S) = (\tau,$
$\tau = \#\{f(i) : i \in S$
T is the *multiset* o

Define "good" to
Chance of good: (

To walk from S to
delete one elt, inse

Non-quantum setu
then inner$\cdot$outer lo
Quantum: σ; then
Take σ to minimiz

ion:
e
er.
:
m
a
of u.

$f(u)$.
or;
ing;
g.
$)^{1/2}$.

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs,
exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm:
nearly 2^n calls to f.
Ambainis, using quantum walk:
$\approx 2^{2n/3}$ calls to f.

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S =$
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for

Define "good" to mean $\tau <$
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor
delete one elt, insert one elt

Non-quantum setup cost σ;
then inner $\cdot$ outer loops $\sigma \cdot (2$
Quantum: σ; then $\sigma^{1/2} \cdot (2$
Take σ to minimize $\sigma + 2^n/$

Extreme example of walk:
Completely unrestricted neighbors.
Recompute f at each step.
Mixes instantly. Same as Grover.

More interesting example:
Ambainis distinctness algorithm.

Say f has 2^n inputs,
exactly one collision $\{p, q\}$.
"Collision": $p \neq q$; $f(p) = f(q)$.
Problem: find this collision.

Generic non-quantum algorithm:
nearly 2^n calls to f.
Ambainis, using quantum walk:
$\approx 2^{2n/3}$ calls to f.

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$,
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$.
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S':
delete one elt, insert one elt.

Non-quantum setup cost σ;
then inner·outer loops $\sigma \cdot (2^n/\sigma)^2$.
Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.
Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

example of walk:
ely unrestricted neighbors.
ute f at each step.
stantly. Same as Grover.

teresting example:
s distinctness algorithm.

as 2^n inputs,
one collision $\{p, q\}$.
n": $p \neq q$; $f(p) = f(q)$.
: find this collision.

non-quantum algorithm:
^{n} calls to f.
s, using quantum walk:
calls to f.

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$,
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$.
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S':
delete one elt, insert one elt.

Non-quantum setup cost σ;
then inner · outer loops $\sigma \cdot (2^n/\sigma)^2$.
Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.
Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some co

Simon u
Shor use
for facto
Can use
"Continu
with car

In all of
naturally
satisfyin
Watch c
and exac

of walk:
ricted neighbors.
ach step.
ame as Grover.

xample:
ness algorithm.

ts,
on $\{p, q\}$.
q; $f(p) = f(q)$.
s collision.

tum algorithm:
f.
uantum walk:

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$,
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$.
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S':
delete one elt, insert one elt.

Non-quantum setup cost σ;
then inner$\cdot$outer loops $\sigma \cdot (2^n/\sigma)^2$.
Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.
Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some common Sh

Simon used additi
Shor used addition
for factorization o
Can use addition i
"Continuous" vers
with careful precis

In all of these algo
naturally find "ran
satisfying $f(u) =$
Watch out for hyp
and exact meaning

ghbors.

rover.

thm.

$f(q)$.

thm:

alk:

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$,
define $\varphi(S) = (\tau, T)$ where
$\tau = \#\{f(i) : i \in S\}$ and
T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$.
Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S':
delete one elt, insert one elt.

Non-quantum setup cost σ;
then inner $\cdot$ outer loops $\sigma \cdot (2^n/\sigma)^2$.
Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.
Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2$
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}$
for factorization or discrete l
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handli

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses o
and exact meaning of "rand

Sketch of Ambainis details:

For $S \subseteq \{\text{inputs}\}$ with $\#S = \sigma$, define $\varphi(S) = (\tau, T)$ where $\tau = \#\{f(i) : i \in S\}$ and T is the *multiset* of $f(i)$ for $i \in S$.

Define "good" to mean $\tau < \sigma$. Chance of good: $(\sigma/2^n)^2$.

To walk from S to neighbor S': delete one elt, insert one elt.

Non-quantum setup cost σ; then inner · outer loops $\sigma \cdot (2^n/\sigma)^2$. Quantum: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$. Take σ to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$. Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$ for factorization or discrete logs. Can use addition in $\mathbf{Z}^n$. "Continuous" version: $\mathbf{R}^n$, with careful precision handling.

In all of these algorithms, naturally find "random" s satisfying $f(u) = f(u + s)$. Watch out for hypotheses on f and exact meaning of "random".

of Ambainis details:

{inputs} with $\#S = \sigma$,

$(S) = (\tau, T)$ where

$f(i) : i \in S\}$ and

multiset of $f(i)$ for $i \in S$.

"good" to mean $\tau < \sigma$.

of good: $(\sigma/2^n)^2$.

from S to neighbor S':

ne elt, insert one elt.

ntum setup cost σ;

er $\cdot$ outer loops $\sigma \cdot (2^n/\sigma)^2$.

m: σ; then $\sigma^{1/2} \cdot (2^n/\sigma)$.

to minimize $\sigma + 2^n/\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if
defined
not nece

Termino
also call
of f und
action.
the "hid
(HSP) is

is details:

with $\#S = \sigma$,
T) where
$S\}$ and
of $f(i)$ for $i \in S$.

mean $\tau < \sigma$.
$(\sigma/2^n)^2$.

neighbor S':
ert one elt.

up cost σ;
oops $\sigma \cdot (2^n/\sigma)^2$.
n $\sigma^{1/2} \cdot (2^n/\sigma)$.
ze $\sigma + 2^n/\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if the functi
defined on a more
not necessarily cor

Terminology: {per
also called the "st
of f under the nat
action. In quantur
the "hidden-subgr
(HSP) is to find th

$= \sigma$,

$i \in S$.

σ.

S':

.

$n/\sigma)^2$.

$n/\sigma)$.

$\sigma^{1/2}$.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if the function f is
defined on a more general g
not necessarily commutative

Terminology: $\{$periods of $f\}$
also called the "stabilizer gr
of f under the natural group
action. In quantum algorith
the "hidden-subgroup proble
(HSP) is to find this group.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u + s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if the function f is
defined on a more general group,
not necessarily commutative?

Terminology: $\{$periods of $f\}$ is
also called the "stabilizer group"
of f under the natural group
action. In quantum algorithms,
the "hidden-subgroup problem"
(HSP) is to find this group.

Some common Shor variants

Simon used addition in $(\mathbf{Z}/2)^n$.
Shor used addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
for factorization or discrete logs.
Can use addition in $\mathbf{Z}^n$.
"Continuous" version: $\mathbf{R}^n$,
with careful precision handling.

In all of these algorithms,
naturally find "random" s
satisfying $f(u) = f(u+s)$.
Watch out for hypotheses on f
and exact meaning of "random".

What if the function f is
defined on a more general group,
not necessarily commutative?

Terminology: $\{$periods of $f\}$ is
also called the "stabilizer group"
of f under the natural group
action. In quantum algorithms,
the "hidden-subgroup problem"
(HSP) is to find this group.

Shor's idea + extra work
handles arbitrary finite groups
with $O(n)$ evaluations of f.
Massive caveat here: also need
huge f-independent computation!

ommon Shor variants

sed addition in $(\mathbf{Z}/2)^n$.
ed addition in $\mathbf{Z}$ or $\mathbf{Z}^2$
orization or discrete logs.
addition in $\mathbf{Z}^n$.
uous" version: $\mathbf{R}^n$,
eful precision handling.

these algorithms,
y find "random" s
g $f(u) = f(u + s)$.
out for hypotheses on f
ct meaning of "random".

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: $\{$periods of $f\}$ is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

Shor's idea + extra work handles arbitrary finite groups with $O(n)$ evaluations of f. Massive caveat here: also need huge f-independent computation!

Kuperbe
reduce t
at some
Total co
but sube
evaluatio

Shor alre
subgrou
For hard
the "hid
find s in
given *tw*
satisfyin

or variants

on in $(\mathbf{Z}/2)^n$.
n in $\mathbf{Z}$ or $\mathbf{Z}^2$
r discrete logs.
n $\mathbf{Z}^n$.
sion: $\mathbf{R}^n$,
ion handling.

orithms,
ndom" s
$f(u+s)$.
potheses on f
g of "random".

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: {periods of f} is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

Shor's idea + extra work handles arbitrary finite groups with $O(n)$ evaluations of f. Massive caveat here: also need huge f-independent computation!

Kuperberg: For di
reduce the extra c
at some cost in f
Total cost is super
but subexponentia
evaluations of f +

Shor already hand
subgroups of the c
For hard cases, Ku
the "hidden-*shift*
find s in a commu
given *two* functior
satisfying $f_1(u) =$

s

$2)^n$.

$\mathbf{Z}^2$

logs.

ng.

n f

om".

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: $\{$periods of $f\}$ is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

Shor's idea $+$ extra work handles arbitrary finite groups with $O(n)$ evaluations of f. Massive caveat here: also need huge f-independent computation!

Kuperberg: For dihedral gro
reduce the extra computatio
at some cost in f evaluation
Total cost is superpolynomia
but subexponential: $2^{O(\sqrt{n})}$
evaluations of f $+$ overhead

Shor already handles some e
subgroups of the dihedral gr
For hard cases, Kuperberg s
the "hidden-*shift* problem":
find s in a commutative gro
given *two* functions f_0, f_1
satisfying $f_1(u) = f_0(u+s)$.

What if the function f is defined on a more general group, not necessarily commutative?

Terminology: $\{$periods of $f\}$ is also called the "stabilizer group" of f under the natural group action. In quantum algorithms, the "hidden-subgroup problem" (HSP) is to find this group.

Shor's idea + extra work handles arbitrary finite groups with $O(n)$ evaluations of f. Massive caveat here: also need huge f-independent computation!

Kuperberg: For dihedral group, reduce the extra computation at some cost in f evaluations. Total cost is superpolynomial but subexponential: $2^{O(\sqrt{n})}$ evaluations of f + overhead.

Shor already handles some easy subgroups of the dihedral group. For hard cases, Kuperberg solves the "hidden-*shift* problem": find s in a commutative group given *two* functions f_0, f_1 satisfying $f_1(u) = f_0(u + s)$.

the function f is
on a more general group,
essarily commutative?

logy: {periods of f} is
ed the "stabilizer group"
ler the natural group
In quantum algorithms,
den-subgroup problem"
s to find this group.

dea + extra work
arbitrary finite groups
n) evaluations of f.
caveat here: also need
ndependent computation!

Kuperberg: For dihedral group,
reduce the extra computation
at some cost in f evaluations.
Total cost is superpolynomial
but subexponential: $2^{O(\sqrt{n})}$
evaluations of f + overhead.

Shor already handles some easy
subgroups of the dihedral group.
For hard cases, Kuperberg solves
the "hidden-*shift* problem":
find s in a commutative group
given *two* functions f_0, f_1
satisfying $f_1(u) = f_0(u+s)$.

The imp

2021.12
to `http`
is encry
by AES-
using a
by the X
with the
by an E
with the
by an R
which in
by an R
which is
SHA-25

on f is
general group,
mmutative?

riods of $f\}$ is
abilizer group"
tural group
m algorithms,
oup problem"
his group.

a work
inite groups
ions of f.
re: also need
nt computation!

Kuperberg: For dihedral group, reduce the extra computation at some cost in f evaluations. Total cost is superpolynomial but subexponential: $2^{O(\sqrt{n})}$ evaluations of f + overhead.

Shor already handles some easy subgroups of the dihedral group. For hard cases, Kuperberg solves the "hidden-*shift* problem": find s in a commutative group given *two* functions f_0, f_1 satisfying $f_1(u) = f_0(u+s)$.

The impact on cry

2021.12: A Firefox
to `https://goog`
is encrypted and a
by AES-128-GCM,
using a key exchar
by the X25519 EC
with the key excha
by an ECDSA-NIS
with the signing k
by an RSA-2048 k
which in turn is ce
by an RSA-4096 k
which is trusted by
SHA-256, SHA-38

Kuperberg: For dihedral group, reduce the extra computation at some cost in f evaluations. Total cost is superpolynomial but subexponential: $2^{O(\sqrt{n})}$ evaluations of f + overhead.

Shor already handles some easy subgroups of the dihedral group. For hard cases, Kuperberg solves the "hidden-*shift* problem": find s in a commutative group given *two* functions f_0, f_1 satisfying $f_1(u) = f_0(u + s)$.

The impact on cryptography

2021.12: A Firefox connecti
to `https://google.com`
is encrypted and authenticat
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH syster
with the key exchange signe
by an ECDSA-NIST-P-256 k
with the signing key certifie
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also app

Kuperberg: For dihedral group, reduce the extra computation at some cost in f evaluations. Total cost is superpolynomial but subexponential: $2^{O(\sqrt{n})}$ evaluations of f + overhead.

Shor already handles some easy subgroups of the dihedral group. For hard cases, Kuperberg solves the "hidden-*shift* problem": find s in a commutative group given *two* functions f_0, f_1 satisfying $f_1(u) = f_0(u + s)$.

The impact on cryptography

2021.12: A Firefox connection to `https://google.com` is encrypted and authenticated by AES-128-GCM, using a key exchanged by the X25519 ECDH system, with the key exchange signed by an ECDSA-NIST-P-256 key, with the signing key certified by an RSA-2048 key, which in turn is certified by an RSA-4096 key, which is trusted by Firefox. SHA-256, SHA-384 also appear.

erg: For dihedral group,
he extra computation
cost in f evaluations.
st is superpolynomial
exponential: $2^{O(\sqrt{n})}$
ons of f + overhead.

eady handles some easy
ps of the dihedral group.
cases, Kuperberg solves
den-*shift* problem":
a commutative group
vo functions f_0, f_1
g $f_1(u) = f_0(u+s)$.

The impact on cryptography

2021.12: A Firefox connection
to `https://google.com`
is encrypted and authenticated
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH system,
with the key exchange signed
by an ECDSA-NIST-P-256 key,
with the signing key certified
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also appear.

Shor's a
ECC in

hedral group,
omputation
evaluations.
rpolynomial
l: $2^{O(\sqrt{n})}$
- overhead.

les some easy
dihedral group.
uperberg solves
problem”:
tative group
ns f_0, f_1
$f_0(u + s)$.

The impact on cryptography

2021.12: A Firefox connection
to `https://google.com`
is encrypted and authenticated
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH system,
with the key exchange signed
by an ECDSA-NIST-P-256 key,
with the signing key certified
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also appear.

Shor’s algorithm b
ECC in polynomia

up,
on
s.
al
.
easy
oup.
olves
up

The impact on cryptography

2021.12: A Firefox connection
to `https://google.com`
is encrypted and authenticated
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH system,
with the key exchange signed
by an ECDSA-NIST-P-256 key,
with the signing key certified
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA
ECC in polynomial time. Pa

The impact on cryptography

2021.12: A Firefox connection to `https://google.com` is encrypted and authenticated by AES-128-GCM, using a key exchanged by the X25519 ECDH system, with the key exchange signed by an ECDSA-NIST-P-256 key, with the signing key certified by an RSA-2048 key, which in turn is certified by an RSA-4096 key, which is trusted by Firefox. SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

The impact on cryptography

2021.12: A Firefox connection
to `https://google.com`
is encrypted and authenticated
by AES-128-GCM,
using a key exchanged
by the X25519 ECDH system,
with the key exchange signed
by an ECDSA-NIST-P-256 key,
with the signing key certified
by an RSA-2048 key,
which in turn is certified
by an RSA-4096 key,
which is trusted by Firefox.
SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

The impact on cryptography

2021.12: A Firefox connection to `https://google.com` is encrypted and authenticated by AES-128-GCM, using a key exchanged by the X25519 ECDH system, with the key exchange signed by an ECDSA-NIST-P-256 key, with the signing key certified by an RSA-2048 key, which in turn is certified by an RSA-4096 key, which is trusted by Firefox. SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

The impact on cryptography

2021.12: A Firefox connection to `https://google.com` is encrypted and authenticated by AES-128-GCM, using a key exchanged by the X25519 ECDH system, with the key exchange signed by an ECDSA-NIST-P-256 key, with the signing key certified by an RSA-2048 key, which in turn is certified by an RSA-4096 key, which is trusted by Firefox. SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

The impact on cryptography

2021.12: A Firefox connection to `https://google.com` is encrypted and authenticated by AES-128-GCM, using a key exchanged by the X25519 ECDH system, with the key exchange signed by an ECDSA-NIST-P-256 key, with the signing key certified by an RSA-2048 key, which in turn is certified by an RSA-4096 key, which is trusted by Firefox. SHA-256, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

act on cryptography

: A Firefox connection
s://google.com
pted and authenticated
128-GCM,
key exchanged
X25519 ECDH system,
key exchange signed
CDSA-NIST-P-256 key,
signing key certified
SA-2048 key,
turn is certified
SA-4096 key,
trusted by Firefox.
6, SHA-384 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

"Is the p
to be a

— 2019
"How to
integers
million r
an impro
algorithm
assumpt
supercon

Most of
correctio
qubits to
qubits in

yptography

x connection
`le.com`
uthenticated
,
nged
EDH system,
ange signed
T-P-256 key,
ey certified
key,
ertified
key,
y Firefox.
4 also appear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

"Is the polynomial
to be a real threat

— 2019 Gidney–E
"How to factor 20
integers in 8 hours
million noisy qubit
an improved versio
algorithm with "pl
assumptions for la
superconducting q

Most of the cost is
correction, using n
qubits to simulate
qubits inside Shor'

y

on

ted

m,

d

key,

d

pear.

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

"Is the polynomial small enc
to be a real threat?"

— 2019 Gidney–Ekerå
"How to factor 2048 bit RS
integers in 8 hours using 20
million noisy qubits" combi
an improved version of Shor
algorithm with "plausible ph
assumptions for large-scale
superconducting qubit platf

Most of the cost is for error
correction, using many impe
qubits to simulate the perfe
qubits inside Shor's algorith

Shor's algorithm breaks RSA and ECC in polynomial time. Panic!

"But nobody has a big enough quantum computer yet!"

— Will large-scale attackers *tell us* that they've built a big enough quantum computer?

Also, leaks show that they're already recording ciphertexts that they'll try to decrypt later.

Also, upgrading everything to post-quantum cryptography won't happen instantaneously.

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

lgorithm breaks RSA and
polynomial time. Panic!

body has a big enough
n computer yet!"

large-scale attackers
hat they've built
ough quantum computer?

ks show that they're
recording ciphertexts
y'll try to decrypt later.

grading everything
quantum cryptography
appen instantaneously.

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

Same pa
million c
log in $\mathbf{F}_p^*$
and $(p -$

(Other p
256-bit e

Useful c
modular
Intel CP
Reasona
compute
overall c
will be 2

reaks RSA and
l time. Panic!

a big enough
r yet!"

attackers
e built
ntum computer?

hat they're
ciphertexts
decrypt later.

verything
cryptography
antaneously.

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

Same paper says 7
million qubits for a
log in $\mathbf{F}_p^*$ if p is a
and $(p-1)/2$ is a

(Other papers: lov
256-bit elliptic-cur

Useful comparison
modular exponent
Intel CPU core is
Reasonable estima
computer will cost
overall cost of qub
will be $2^{40}\times$ cost

A and
nic!

gh

puter?

e
s
ter.

hy
ly.

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

Same paper says 7 hours wi
million qubits for a big discr
log in $\mathbf{F}_p^*$ if p is a 2048-bit p
and $(p-1)/2$ is also prime.

(Other papers: lower costs f
256-bit elliptic-curve discrete

Useful comparison: *non-qua*
modular exponentiation on a
Intel CPU core is $>2^{20}\times$ fas
Reasonable estimates: quan
computer will cost $2^{20}\times$ mo
overall cost of qubit operatio
will be $2^{40}\times$ cost of bit oper

"Is the polynomial small enough to be a real threat?"

— 2019 Gidney–Ekerå "How to factor 2048 bit RSA integers in 8 hours using 20 million noisy qubits" combines an improved version of Shor's algorithm with "plausible physical assumptions for large-scale superconducting qubit platforms".

Most of the cost is for error correction, using many imperfect qubits to simulate the perfect qubits inside Shor's algorithm.

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

polynomial small enough
real threat?"

Gidney–Ekerå
o factor 2048 bit RSA
in 8 hours using 20
noisy qubits" combines
oved version of Shor's
m with "plausible physical
ions for large-scale
nducting qubit platforms".

the cost is for error
on, using many imperfect
o simulate the perfect
nside Shor's algorithm.

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

Some re
• Lower-
• Less n
• Better
• Better
• Better

Beyond
each alg
quantum
CCNOT
reversibi
with hig
error-cor
size of c

small enough
?"

kerå
48 bit RSA
s using 20
s" combines
on of Shor's
ausible physical
rge-scale
ubit platforms".

s for error
many imperfect
the perfect
's algorithm.

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

Some reasons 2^{40}
- Lower-cost qubit
- Less noise in qu
- Better qubit con
- Better error-corr
- Better reversibili

Beyond modular e
each algorithm ne
quantum/non-qua
CCNOT costs >1
reversibility conve
with high-level alg
error-correction co
size of computatio

ough

A

nes

's

ysical

orms".

erfect

ct

m.

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

Some reasons 2^{40} can impro

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction me
- Better reversibility convers

Beyond modular exponentia each algorithm needs analys quantum/non-quantum cost CCNOT costs $>100\times$ CNO reversibility conversions inte with high-level algorithm de error-correction cost depend size of computation; and so

Same paper says 7 hours with 26 million qubits for a big discrete log in $\mathbf{F}_p^*$ if p is a 2048-bit prime and $(p-1)/2$ is also prime.

(Other papers: lower costs for 256-bit elliptic-curve discrete log.)

Useful comparison: *non-quantum* modular exponentiation on an Intel CPU core is $>2^{20}\times$ faster. Reasonable estimates: quantum computer will cost $2^{20}\times$ more; overall cost of qubit operation will be $2^{40}\times$ cost of bit operation.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

aper says 7 hours with 26
qubits for a big discrete
$^*_p$ if p is a 2048-bit prime
– 1)/2 is also prime.

papers: lower costs for
elliptic-curve discrete log.)

omparison: *non-quantum*
exponentiation on an
U core is $>2^{20}\times$ faster.
ble estimates: quantum
er will cost $2^{20}\times$ more;
ost of qubit operation
$2^{40}\times$ cost of bit operation.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further

Typical s
Guess th
see if gu
to <HTML
If not, tr

Non-qua
in 2^{127} g
which so
for most
(Bitcoin

7 hours with 26
a big discrete
2048-bit prime
lso prime.

wer costs for
ve discrete log.)

: *non-quantum*
iation on an
$>2^{20}\times$ faster.
tes: quantum
: $2^{20}\times$ more;
bit operation
of bit operation.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further impact: G

Typical symmetric
Guess the secret A
see if guess decryp
to `<HTML><HEAD><m`
If not, try further

Non-quantum atta
in 2^{127} guesses on
which sounds too
for most people to
(Bitcoin: $\approx 2^{92}$ ha

th 26
ete
prime

for
e log.)

ntum
an
ster.
tum
re;
on
ration.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further impact: Grover

Typical symmetric-crypto at
Guess the secret AES-128 ke
see if guess decrypts ciphert
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succee
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry ab
(Bitcoin: $\approx 2^{92}$ hashes/year.

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further impact: Grover

Typical symmetric-crypto attack: Guess the secret AES-128 key, see if guess decrypts ciphertext to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds in 2^{127} guesses on average, which sounds too expensive for most people to worry about. (Bitcoin: $\approx 2^{92}$ hashes/year.)

Some reasons 2^{40} can improve:

- Lower-cost qubits.
- Less noise in qubits.
- Better qubit connectivity.
- Better error-correction methods.
- Better reversibility conversions.

Beyond modular exponentiation: each algorithm needs analysis of quantum/non-quantum cost ratio. CCNOT costs $>100\times$ CNOT; reversibility conversions interact with high-level algorithm details; error-correction cost depends on size of computation; and so on.

Further impact: Grover

Typical symmetric-crypto attack: Guess the secret AES-128 key, see if guess decrypts ciphertext to `<HTML><HEAD><met...` If not, try further guesses.

Non-quantum attack succeeds in 2^{127} guesses on average, which sounds too expensive for most people to worry about. (Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum evaluations of AES. Panic!

asons 2^{40} can improve:
-cost qubits.
oise in qubits.
qubit connectivity.
error-correction methods.
reversibility conversions.

modular exponentiation:
orithm needs analysis of
n/non-quantum cost ratio.
costs $>100\times$ CNOT;
lity conversions interact
h-level algorithm details;
rrection cost depends on
omputation; and so on.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantur
$\approx 2^{15}$ qu
Similar c
Attack c

can improve:
ts.
bits.
nectivity.
rection methods.
ity conversions.

xponentiation:
eds analysis of
ntum cost ratio.
00$\times$ CNOT;
rsions interact
orithm details;
st depends on
n; and so on.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantum AES eva
$\approx 2^{15}$ qubit operat
Similar cost to 2^{55}
Attack costs $\approx 2^{11}$

ove:

thods.

sions.

tion:

is of

ratio.

T;

ract

tails;

s on

on.

Further impact: Grover

Typical symmetric-crypto attack: Guess the secret AES-128 key, see if guess decrypts ciphertext to <HTML><HEAD><met...
If not, try further guesses.

Non-quantum attack succeeds in 2^{127} guesses on average, which sounds too expensive for most people to worry about. (Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum evaluations of AES. Panic!

Quantum AES evaluation: $\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit opera
Attack costs $\approx 2^{119}$ bit oper

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key, see if guess decrypts ciphertext to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds in 2^{127} guesses on average, which sounds too expensive for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum evaluations of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds = 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

Further impact: Grover

Typical symmetric-crypto attack:
Guess the secret AES-128 key,
see if guess decrypts ciphertext
to `<HTML><HEAD><met...`
If not, try further guesses.

Non-quantum attack succeeds
in 2^{127} guesses on average,
which sounds too expensive
for most people to worry about.
(Bitcoin: $\approx 2^{92}$ hashes/year.)

Grover takes only 2^{64} quantum
evaluations of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

impact: Grover

symmetric-crypto attack:
he secret AES-128 key,
ess decrypts ciphertext
L><HEAD><met...
ry further guesses.

ntum attack succeeds
guesses on average,
ounds too expensive
people to worry about.
: $\approx 2^{92}$ hashes/year.)

akes only 2^{64} quantum
ons of AES. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still
than nor
under re
re quant
but muc
than Sho

Many co
that AES

rover

-crypto attack:
ES-128 key,
ts ciphertext
`met...`
guesses.

ack succeeds
average,
expensive
worry about.
shes/year.)

2^{64} quantum
S. Panic!

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still be lower
than non-quantum
under reasonable a
re quantum-compu
but much more ex
than Shor RSA-20

Many commentato
that AES-128 is sa

tack:
ey,
ext
ds
out.
)
um

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still be lower cost
than non-quantum AES atta
under reasonable assumptio
re quantum-computer progre
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclu
that AES-128 is safe.

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

Quantum AES evaluation:
$\approx 2^{15}$ qubit operations.
Similar cost to 2^{55} bit operations.
Attack costs $\approx 2^{119}$ bit operations.

Also, Grover speedup
comes from *serial* iterations.
2^{64} nanoseconds $=$ 585 years,
and 1ns iterations won't be easy.

To run $2^{10}\times$ faster,
need 2^{20} quantum computers:
$\approx 2^{129}$ bit operations.

To run $2^{20}\times$ faster,
need 2^{40} quantum computers:
$\approx 2^{139}$ bit operations.

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

However, AES-128 exposes many
protocols to multi-target attacks
that are already feasible today.
So use AES-256. (Or ChaCha20:
bigger security margin, no timing
attacks, no block-size attacks.)

m AES evaluation:
bit operations.
cost to 2^{55} bit operations.
costs $\approx 2^{119}$ bit operations.

over speedup
rom *serial* iterations.
oseconds = 585 years,
iterations won't be easy.

$2^{10}\times$ faster,
quantum computers:
it operations.

$2^{20}\times$ faster,
quantum computers:
it operations.

Can still be lower cost than non-quantum AES attack under reasonable assumptions re quantum-computer progress, but much more expensive than Shor RSA-2048 attack.

Many commentators conclude that AES-128 is safe.

However, AES-128 exposes many protocols to multi-target attacks that are already feasible today. So use AES-256. (Or ChaCha20: bigger security margin, no timing attacks, no block-size attacks.)

Every Gr
runs into
How ma
willing t
How ma
can be c
for the t
Does thi
qubit-op

aluation:
ions.
5 bit operations.
9 bit operations.

dup
iterations.
= 585 years,
won't be easy.

r,
computers:
ons.

r,
computers:
ons.

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

However, AES-128 exposes many
protocols to multi-target attacks
that are already feasible today.
So use AES-256. (Or ChaCha20:
bigger security margin, no timing
attacks, no block-size attacks.)

Every Grover appli
runs into the same
How many years is
willing to wait for
How many *serial* i
can be carried out
for the target func
Does this outweigh
qubit-op cost and

tions.
ations.
s,
easy.
s:
s:

Can still be lower cost than non-quantum AES attack under reasonable assumptions re quantum-computer progress, but much more expensive than Shor RSA-2048 attack.

Many commentators conclude that AES-128 is safe.

However, AES-128 exposes many protocols to multi-target attacks that are already feasible today. So use AES-256. (Or ChaCha20: bigger security margin, no timing attacks, no block-size attacks.)

Every Grover application runs into the same question
How many years is the user willing to wait for results?
How many *serial* iterations can be carried out in that ti
for the target function f?
Does this outweigh ratio bet
qubit-op cost and bit-op cos

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

However, AES-128 exposes many
protocols to multi-target attacks
that are already feasible today.
So use AES-256. (Or ChaCha20:
bigger security margin, no timing
attacks, no block-size attacks.)

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

Can still be lower cost
than non-quantum AES attack
under reasonable assumptions
re quantum-computer progress,
but much more expensive
than Shor RSA-2048 attack.

Many commentators conclude
that AES-128 is safe.

However, AES-128 exposes many
protocols to multi-target attacks
that are already feasible today.
So use AES-256. (Or ChaCha20:
bigger security margin, no timing
attacks, no block-size attacks.)

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

For cryptographic risk
management, should presume
some Grover speedup depending
on quantum-computer progress,
but have to account for costs of
quantum evaluation of f.

be lower cost
n-quantum AES attack
asonable assumptions
tum-computer progress,
h more expensive
or RSA-2048 attack.

ommentators conclude
S-128 is safe.

r, AES-128 exposes many
s to multi-target attacks
already feasible today.
AES-256. (Or ChaCha20:
ecurity margin, no timing
no block-size attacks.)

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

For cryptographic risk
management, should presume
some Grover speedup depending
on quantum-computer progress,
but have to account for costs of
quantum evaluation of f.

For man
(and qua
claims o
in the lit
underest

Example
finds SH
evaluatio
finds SH
evaluatio
accesses
The liter
physicall
where qu

cost
n AES attack
assumptions
uter progress,
pensive
48 attack.

ors conclude
afe.

3 exposes many
-target attacks
easible today.
(Or ChaCha20:
rgin, no timing
size attacks.)

Every Grover application runs into the same questions. How many years is the user willing to wait for results? How many *serial* iterations can be carried out in that time for the target function f? Does this outweigh ratio between qubit-op cost and bit-op cost?

For cryptographic risk management, should presume some Grover speedup depending on quantum-computer progress, but have to account for costs of quantum evaluation of f.

For many applicat
(and quantum-wal
claims of quantum
in the literature re
underestimating th

Example: Non-qua
finds SHA-256 col
evaluations. Quan
finds SHA-256 col
evaluations *plus* 2
accesses to 2^{85} me
The literature doe
physically plausibl
where quantum al

ack
ns
ess,
.
de

many
acks
ay.
ha20:
ming
ks.)

Every Grover application
runs into the same questions.
How many years is the user
willing to wait for results?
How many *serial* iterations
can be carried out in that time
for the target function f?
Does this outweigh ratio between
qubit-op cost and bit-op cost?

For cryptographic risk
management, should presume
some Grover speedup depending
on quantum-computer progress,
but have to account for costs of
quantum evaluation of f.

For many applications of Gr
(and quantum-walk algorith
claims of quantum speedups
in the literature rely criticall
underestimating the cost of

Example: Non-quantum alg
finds SHA-256 collision in 2
evaluations. Quantum algor
finds SHA-256 collision in 2
evaluations *plus* 2^{85} random
accesses to 2^{85} memory loca
The literature does not state
physically plausible cost mod
where quantum algorithm w

Every Grover application runs into the same questions. How many years is the user willing to wait for results? How many *serial* iterations can be carried out in that time for the target function f? Does this outweigh ratio between qubit-op cost and bit-op cost?

For cryptographic risk management, should presume some Grover speedup depending on quantum-computer progress, but have to account for costs of quantum evaluation of f.

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

rover application
o the same questions.
ny years is the user
o wait for results?
ny *serial* iterations
carried out in that time
arget function f?
is outweigh ratio between
o cost and bit-op cost?

tographic risk
ment, should presume
rover speedup depending
tum-computer progress,
e to account for costs of
n evaluation of f.

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

Post-qua

What do
against c

2003: C
"post-qu

2006, 20
2014, 20
2020, 20
PQCrypt

2015: N
2016: N
Post-Qu
Standard

ication
e questions.
s the user
results?
terations
in that time
tion f?
h ratio between
bit-op cost?

risk
uld presume
dup depending
uter progress,
nt for costs of
on of f.

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

Post-quantum cry

What do cryptogra
against quantum c

2003: Coined the
"post-quantum cry

2006, 2008, 2010,
2014, 2016, 2017,
2020, 2021, . . . :
PQCrypto confere

2015: NSA issued
2016: NIST annou
Post-Quantum Cry
Standardization Pr

s.

me

tween
st?

ne
ding
ress,
ts of

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

Post-quantum cryptography

What do cryptographers do against quantum computers

2003: Coined the term "post-quantum cryptograph

2006, 2008, 2010, 2011, 201
2014, 2016, 2017, 2018, 201
2020, 2021, . . . :
PQCrypto conferences.

2015: NSA issued statemen
2016: NIST announced
Post-Quantum Cryptograph
Standardization Project.

For many applications of Grover (and quantum-walk algorithms), claims of quantum speedups in the literature rely critically on underestimating the cost of f.

Example: Non-quantum algorithm finds SHA-256 collision in 2^{128} evaluations. Quantum algorithm finds SHA-256 collision in 2^{85} evaluations *plus* 2^{85} random accesses to 2^{85} memory locations. The literature does not state a physically plausible cost model where quantum algorithm wins.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

y applications of Grover
antum-walk algorithms),
f quantum speedups
terature rely critically on
timating the cost of f.

e: Non-quantum algorithm
IA-256 collision in 2^{128}
ons. Quantum algorithm
IA-256 collision in 2^{85}
ons *plus* 2^{85} random
to 2^{85} memory locations.
rature does not state a
ly plausible cost model
uantum algorithm wins.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: N
69 comp

ions of Grover
k algorithms),
speedups
ly critically on
he cost of f.

antum algorithm
lision in 2^{128}
tum algorithm
lision in 2^{85}
^{85} random
emory locations.
s not state a
e cost model
gorithm wins.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST receiv
69 complete subm

over
ms),
s
y on
f.

orithm
128

ithm
85

n
ations.
e a
del
ins.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST received and po
69 complete submissions.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST received and posted 69 complete submissions.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Post-quantum cryptography

What do cryptographers do against quantum computers?

2003: Coined the term "post-quantum cryptography".

2006, 2008, 2010, 2011, 2013, 2014, 2016, 2017, 2018, 2019, 2020, 2021, . . . : PQCrypto conferences.

2015: NSA issued statement.
2016: NIST announced Post-Quantum Cryptography Standardization Project.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven't been found yet.

antum cryptography

o cryptographers do
quantum computers?

oined the term
uantum cryptography".

008, 2010, 2011, 2013,
016, 2017, 2018, 2019,
021, . . . :
to conferences.

SA issued statement.
IST announced
antum Cryptography
dization Project.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven't been found yet.

Major di
quantum

1. "Sma
Main de
Try to fi
that can
searches

Some st
built this
key sear
for SPH
for Class
XL for M
enumera

ptography

aphers do
computers?

term
yptography”.

2011, 2013,
2018, 2019,

nces.

statement.
unced
yptography
roject.

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven’t been found yet.

Major directions s
quantum cryptana

1. “Small” Grove
Main design strate
Try to find attack
that can be viewe
searches for “goo

Some state-of-the-
built this way: qu
key search; quant
for SPHINCS+; q
for Classic McElie
XL for MQ syste
enumeration for la

?

y”.

13,

19,

t.

y

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven’t been found yet.

Major directions so far in *quantum* cryptanalysis of PQ

1. “Small” Grover applicatio
Main design strategy:
Try to find attack componen
that can be viewed as huge
searches for “good” objects.

Some state-of-the-art attack
built this way: quantum AE
key search; quantum preima
for SPHINCS+; quantum IS
for Classic McEliece; quantu
XL for MQ systems; quantu
enumeration for lattice syste

2017: NIST received and posted 69 complete submissions.

Almost all submissions have faster attacks known today *even without quantum computers*. About half have been shown to not meet their security claims. New attack advances are continuing to appear in 2021.

Much worse status than previous cryptographic competitions. Cryptanalysts are overloaded. Presumably many attacks haven't been found yet.

Major directions so far in *quantum* cryptanalysis of PQC:

1. "Small" Grover applications. Main design strategy: Try to find attack components that can be viewed as huge searches for "good" objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

IST received and posted
plete submissions.

all submissions have
tacks known today
thout quantum computers.
alf have been shown to
t their security claims.
ack advances are
ng to appear in 2021.

orse status than previous
raphic competitions.
alysts are overloaded.
ably many attacks
been found yet.

Major directions so far in *quantum* cryptanalysis of PQC:

1. "Small" Grover applications. Main design strategy: Try to find attack components that can be viewed as huge searches for "good" objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

2. "Big"
quantum
Main de
find atta
be viewe

Some st
built thi
costs ma
quantum
quantum
quantum
approach
quantum
for lattic

ved and posted
issions.

sions have
wn today
ntum computers.
een shown to
urity claims.
ces are
ear in 2021.

s than previous
npetitions.
overloaded.
attacks
d yet.

Major directions so far in *quantum* cryptanalysis of PQC:

1. "Small" Grover applications.
Main design strategy:
Try to find attack components that can be viewed as huge searches for "good" objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

2. "Big" Grover a
quantum walks, et
Main design strate
find attack compo
be viewed as collis

Some state-of-the-
built this way, *ass*
costs magically dis
quantum collisions
quantum claw-find
quantum sieving (
approach) for latti
quantum combina
for lattice systems

osted

outers.
n to
ns.

1.

vious

d.

Major directions so far in *quantum* cryptanalysis of PQC:

1. “Small” Grover applications. Main design strategy: Try to find attack components that can be viewed as huge searches for “good” objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

2. “Big” Grover application
quantum walks, etc.
Main design strategy: Try to
find attack components that
be viewed as collision search

Some state-of-the-art attack
built this way, *assuming me*
costs magically disappear:
quantum collisions for SHA-
quantum claw-finding for SI
quantum sieving (and anoth
approach) for lattice system
quantum combinatorial atta
for lattice systems.

Major directions so far in *quantum* cryptanalysis of PQC:

1. "Small" Grover applications.
Main design strategy:
Try to find attack components that can be viewed as huge searches for "good" objects.

Some state-of-the-art attacks built this way: quantum AES key search; quantum preimages for SPHINCS+; quantum ISD for Classic McEliece; quantum XL for MQ systems; quantum enumeration for lattice systems.

2. "Big" Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

irections so far in
n cryptanalysis of PQC:

all" Grover applications.
sign strategy:
nd attack components
be viewed as huge
for "good" objects.

ate-of-the-art attacks
s way: quantum AES
ch; quantum preimages
INCS+; quantum ISD
sic McEliece; quantum
MQ systems; quantum
tion for lattice systems.

2. "Big" Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

3. Kupe
and opti
example
isogeny-
non-inte

(This su
prompte
SIKE. S
CRS/CS
security
attacks,
SIKE als
avenues
non-inte

o far in
lysis of PQC:

r applications.
egy:
components
d as huge
d" objects.

-art attacks
antum AES
um preimages
uantum ISD
ce; quantum
ns; quantum
ttice systems.

2. "Big" Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

3. Kuperberg appl
and optimizations.
example: attackin
isogeny-based syst
non-interactive key

(This subexponent
prompted the deve
SIKE. SIKE is sma
CRS/CSIDH for *s*
security levels aga
attacks, but cutof
SIKE also opens u
avenues and doesn
non-interactive key

QC:

ons.

nts

ks

S

ges

D

um

m

ems.

2. “Big” Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

3. Kuperberg applications and optimizations. Interestin example: attacking CRS/CS isogeny-based systems for sm non-interactive key exchange

(This subexponential CRS a prompted the development o SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently* security levels against know attacks, but cutoff is unclea SIKE also opens up new att avenues and doesn't provide non-interactive key exchange

2. "Big" Grover applications, quantum walks, etc.
Main design strategy: Try to find attack components that can be viewed as collision searches.

Some state-of-the-art attacks built this way, *assuming memory costs magically disappear*: quantum collisions for SHA-256; quantum claw-finding for SIKE; quantum sieving (and another approach) for lattice systems; quantum combinatorial attacks for lattice systems.

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

' Grover applications,
n walks, etc.
sign strategy: Try to
ack components that can
ed as collision searches.

ate-of-the-art attacks
s way, *assuming memory
agically disappear*:
n collisions for SHA-256;
n claw-finding for SIKE;
n sieving (and another
n) for lattice systems;
n combinatorial attacks
ce systems.

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

4. Shor
Interesti
logarithr
number
further t
polynom
usual "c
of Gentr
homomo
ideal lat
lattice-b

Latest d
see recer
against l

pplications,
tc.
egy: Try to
nents that can
sion searches.

-art attacks
uming memory
sappear:
s for SHA-256;
ling for SIKE;
and another
ce systems;
torial attacks
.

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

4. Shor applicatio
Interesting exampl
logarithms in grou
number fields, con
further techniques
polynomial-time a
usual "cyclotomic
of Gentry STOC 2
homomorphic encr
ideal lattices" and
lattice-based crypt

Latest developmen
see recent talk on
against Ideal-SVP.

s,

o

t can

es.

ks

mory

256;

KE;

er

s;

cks

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

4. Shor applications.

Interesting example: discret
logarithms in groups related
number fields, combined wit
further techniques, led to a
polynomial-time attack brea
usual "cyclotomic $h^+ = 1$"
of Gentry STOC 2009 "Fully
homomorphic encryption usi
ideal lattices" and some new
lattice-based cryptosystems.

Latest developments:
see recent talk on S-unit att
against Ideal-SVP.

3. Kuperberg applications and optimizations. Interesting example: attacking CRS/CSIDH, isogeny-based systems for small non-interactive key exchange.

(This subexponential CRS attack prompted the development of SIKE. SIKE is smaller than CRS/CSIDH for *sufficiently large* security levels against known attacks, but cutoff is unclear. SIKE also opens up new attack avenues and doesn't provide non-interactive key exchange.)

4. Shor applications. Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments: see recent talk on S-unit attacks against Ideal-SVP.

rberg applications
mizations. Interesting
: attacking CRS/CSIDH,
based systems for small
ractive key exchange.

bexponential CRS attack
d the development of
IKE is smaller than
SIDH for *sufficiently large*
levels against known
but cutoff is unclear.
so opens up new attack
and doesn't provide
ractive key exchange.)

4. Shor applications.
Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments:
see recent talk on S-unit attacks against Ideal-SVP.

5. New
attacks.
"Quantu
of avera
via filter

lications
. Interesting
g CRS/CSIDH,
ems for small
y exchange.

tial CRS attack
elopment of
aller than
ufficiently large
inst known
f is unclear.
p new attack
n't provide
y exchange.)

4. Shor applications.
Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments:
see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for c
attacks. Recent ex
"Quantum algorith
of average-case lat
via filtering".

ng
SIDH,
mall
e.

ttack
of

large
n
r.
ack
e
e.)

4. Shor applications.

Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^{+} = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments: see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for va
of average-case lattice probl
via filtering".

4. Shor applications.

Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments: see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

4. Shor applications.

Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments: see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

6. Analyzing and optimizing costs of all of these algorithms in much more detail.

4. Shor applications.
Interesting example: discrete logarithms in groups related to number fields, combined with further techniques, led to a polynomial-time attack breaking usual "cyclotomic $h^+ = 1$" case of Gentry STOC 2009 "Fully homomorphic encryption using ideal lattices" and some newer lattice-based cryptosystems.

Latest developments: see recent talk on S-unit attacks against Ideal-SVP.

5. New ideas for quantum attacks. Recent example: "Quantum algorithms for variants of average-case lattice problems via filtering".

6. Analyzing and optimizing costs of all of these algorithms in much more detail.

7. Changing cryptosystems to enable attacks: e.g. "Please use your secret key on a quantum computer to decrypt the following superposition of ciphertexts."