Quantum walks

Daniel J. Bernstein
University of Illinois at Chicago

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

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

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
0.791 . . .	0.462 . . .	MQ
0.290 . . .	0.241 . . .	subset sum

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

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

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

m walks

. Bernstein
ty of Illinois at Chicago

g on quantum walks
gorithm-design tool:
ver's algorithm.
bainis's algorithm.

o study quantum walks
n more general graphs.
ilds, 2009 Lovett–
-Everitt–Trevers–Kendon:
v, e.g., Shor's algorithm
tum walk on Shor graph.

Examples of applications to crypto

Minimum asymptotic ops known,
assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
0.791 . . .	0.462 . . .	MQ
0.290 . . .	0.241 . . .	subset sum

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

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

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

"McEliec
length (
dimensio
decode (

n
is at Chicago

---

tum walks
esign tool:
ithm.
gorithm.

antum walks
neral graphs.
Lovett–
revers–Kendon:
or's algorithm
on Shor graph.

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

"McEliece": in lin
length $(1 + o(1))$
dimension $(R + o$
decode $(1 - R + o$

ago

lks
ns.

ndon:
thm
raph.

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

"McEliece": in linear code o
length $(1 + o(1))n \log_2 n$ an
dimension $(R + o(1))n \log_2$
decode $(1 - R + o(1))n$ erro

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

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

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

"McEliece": in linear code of length $(1 + o(1))n \log_2 n$ and dimension $(R + o(1))n \log_2 n$, decode $(1 - R + o(1))n$ errors. $\rho = (1 - R)\log_2(1/(1 - R))$: 1962 Prange.

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

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

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

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

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

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

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

"MQ": solve system of n deg-2 equations in n variables over $\mathbf{F}_2$. 0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.

Examples of applications to crypto

Minimum asymptotic ops known, assuming plausible heuristics:

pre-q	post-q	problem
1	0.5	cipher
ρ	$\rho/2$	McEliece
$0.791\ldots$	$0.462\ldots$	MQ
$0.290\ldots$	$0.241\ldots$	subset sum

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

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

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

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

"MQ": solve system of n deg-2 equations in n variables over $\mathbf{F}_2$.
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.
0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

es of applications to crypto

m asymptotic ops known,
g plausible heuristics:

	post-q	problem
	0.5	cipher
	$\rho/2$	McEliece
.	$0.462\ldots$	MQ
.	$0.241\ldots$	subset sum

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

” e: as $n \to \infty$, $2^{(e+o(1))n}$
uantum ops.

”: find n-bit cipher key.
96 Grover.

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

“MQ”: solve system of n deg-2 equations in n variables over $\mathbf{F}_2$.
0.791 (modulo calculation errors):
2004 Yang–Chen–Courtois.
0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

“Subset
find $S \subseteq$
$x_1, x_2, \ldots$
and $\sum_{i\in}$

cations to crypto

otic ops known,

e heuristics:

	problem
	cipher
	McEliece
.	MQ
.	subset sum

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

um ops.

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

ps.

bit cipher key.

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

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

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

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

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

0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

"Subset sum" ("h

find $S \subseteq \{1, 2, \ldots$

$x_1, x_2, \ldots, x_n \in \{0$

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

crypto

own,

s:

e

um

$o(1))n$

$+o(1))n$

key.

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

"MQ": solve system of n deg-2 equations in n variables over $\mathbf{F}_2$.
0.791 (modulo calculation errors):
2004 Yang–Chen–Courtois.
0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

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

"MQ": solve system of n deg-2 equations in n variables over $\mathbf{F}_2$.
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.
0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

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

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

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

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

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

0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

0.5: easy.

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

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

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

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

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

0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

0.5: easy.

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

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

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

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

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

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

0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

"MQ": solve system of n deg-2 equations in n variables over $\mathbf{F}_2$.
0.791 (modulo calculation errors): 2004 Yang–Chen–Courtois.
0.462: 2017 Bernstein–Yang (via Grover), independently 2017 Faugère–Horan–Kahrobaei–Kaplan–Kashefi–Perret.

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

ce": in linear code of
$1 + o(1))n \log_2 n$ and
on $(R + o(1))n \log_2 n$,
$(1 - R + o(1))n$ errors.
$- R) \log_2(1/(1 - R))$:
ange.
09 Bernstein (via Grover).

solve system of n deg-2
ns in n variables over $\mathbf{F}_2$.
nodulo calculation errors):
ng–Chen–Courtois.
2017 Bernstein–Yang
ver), independently 2017
–Horan–Kahrobaei–
Kashefi–Perret.

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

Grover's

Assume:
has $f(s)$

Traditio
compute
hope to
Success
until #i

ear code of
$n \log_2 n$ and
$(1))n \log_2 n$,
$o(1))n$ errors.
$1/(1-R))$:

ein (via Grover).

m of n deg-2
iables over $\mathbf{F}_2$.
culation errors):
Courtois.
stein–Yang
pendently 2017
ahrobaei–
erret.

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

Grover's algorithm

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

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

of
nd
n,
ors.
):

over).

g-2
$\mathbf{F}_2$.
rrors):

g
2017

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

Grover's algorithm

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

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

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

Grover's algorithm

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

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

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

0.5: easy.

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

0.291: 2011 Becker–Coron–Joux.

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

Grover's algorithm

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

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

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

sum" ("hard" case):

$\in \{1, 2, \ldots, n\}$ given

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

$_{\in S} x_i$.

y.

2010 Howgrave-Graham–

laimed 0.311; error

ed by May–Meurer.

2011 Becker–Coron–Joux.

2013 Bernstein–Jeffery–

Meurer, using HGJ and

n walks (not just Grover).

Grover's algorithm

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

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

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

Start fro

a over q

ard" case):
, $n\}$ given
$, 1, \ldots, 2^n - 1\}$

grave-Graham–
311; error
–Meurer.

er–Coron–Joux.

stein–Jeffery–
ing HGJ and
ot just Grover).

Grover's algorithm

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

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

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

Start from uniform
a over $q \in \{0,1\}^n$

):

$-1\}$

am–

Joux.

ry–
nd
over).

Grover’s algorithm

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

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

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

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

Grover's algorithm

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

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

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

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

Grover's algorithm

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

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

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

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

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

Grover's algorithm

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

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

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

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

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

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

Grover's algorithm

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

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

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

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

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

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

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

Grover's algorithm

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

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

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

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

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

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

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

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

algorithm

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

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

algorithm takes only $2^{n/2}$
e computations of f.
y: reversibility overhead
enough that this easily
all sufficiently large n.

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

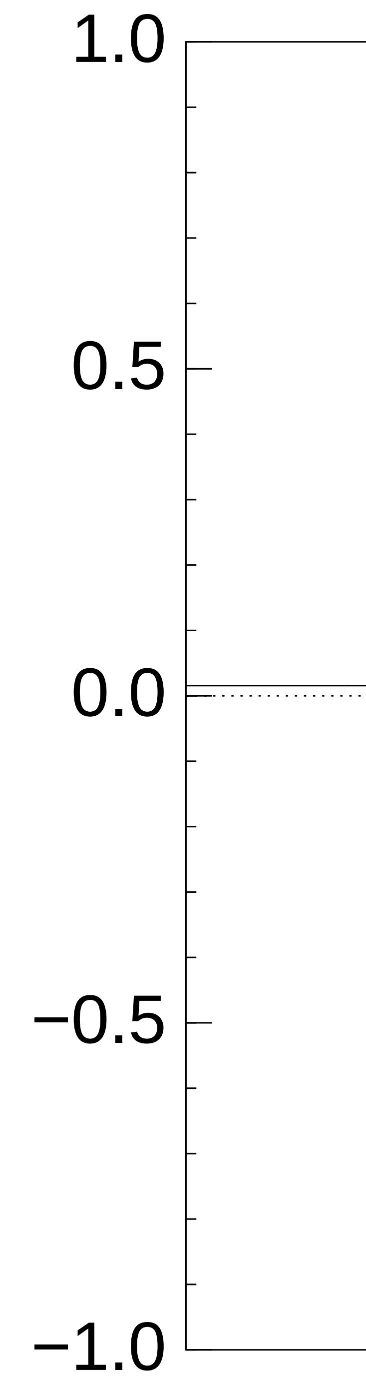
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normaliz
for an ex
after 0 s



n

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

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

n takes only $2^{n/2}$
ations of f.
ility overhead
at this easily
ently large n.

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

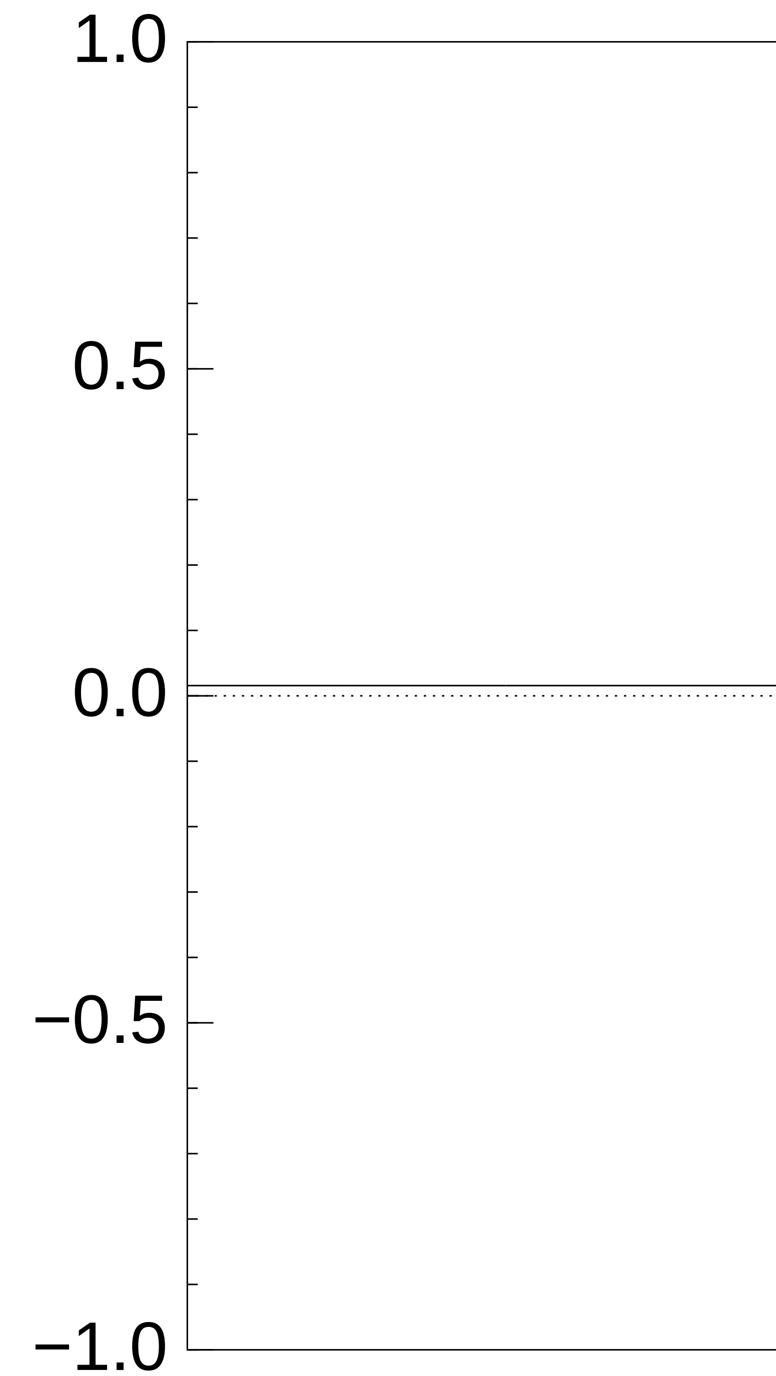
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph
for an example wit
after 0 steps:



l s:

ow

?.

y $2^{n/2}$

?.

ead

sily

n.

5

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

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

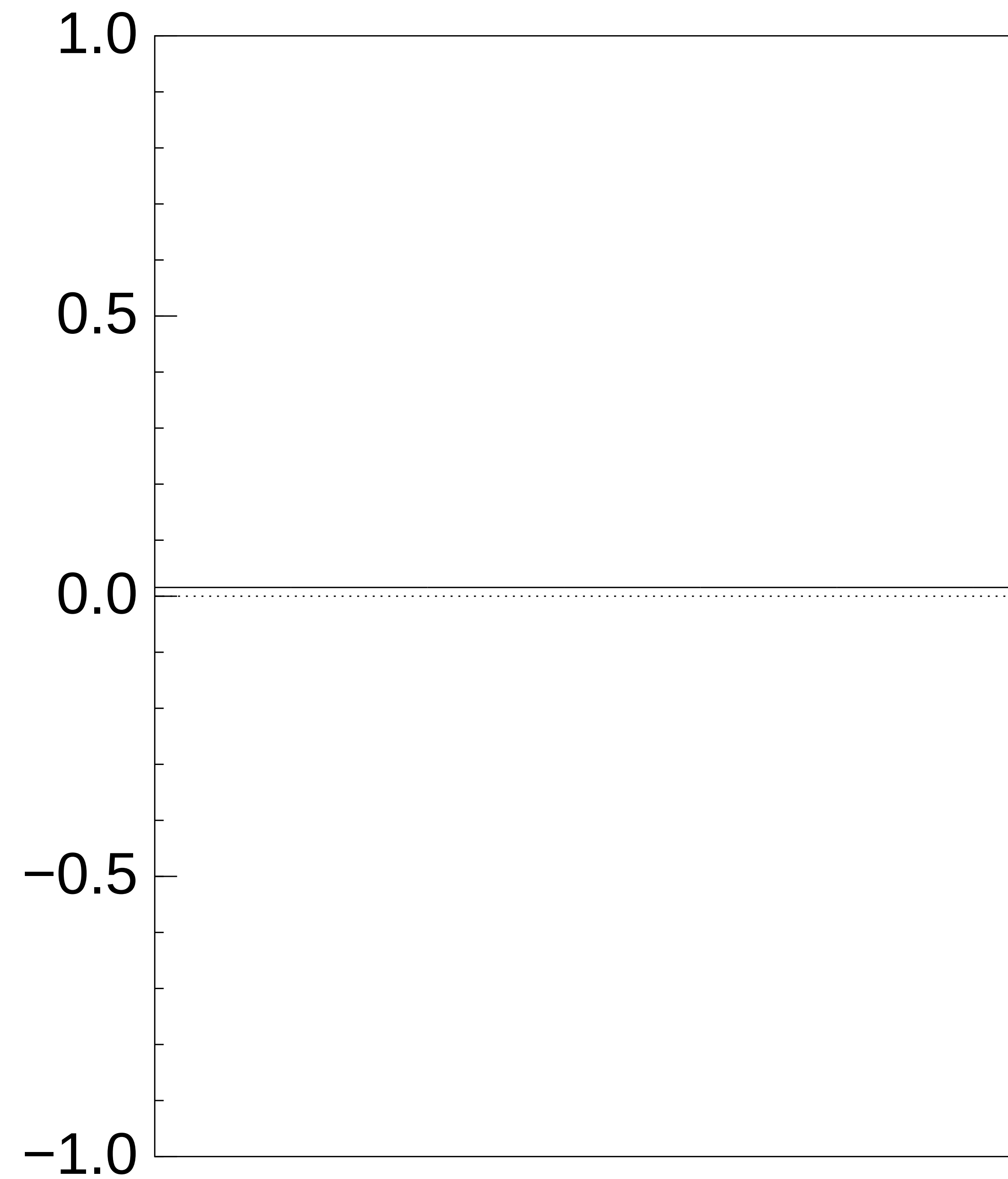
Step 2: "Grover diffusion".
Negate a around its average.
This is also fast.

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

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

6

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



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

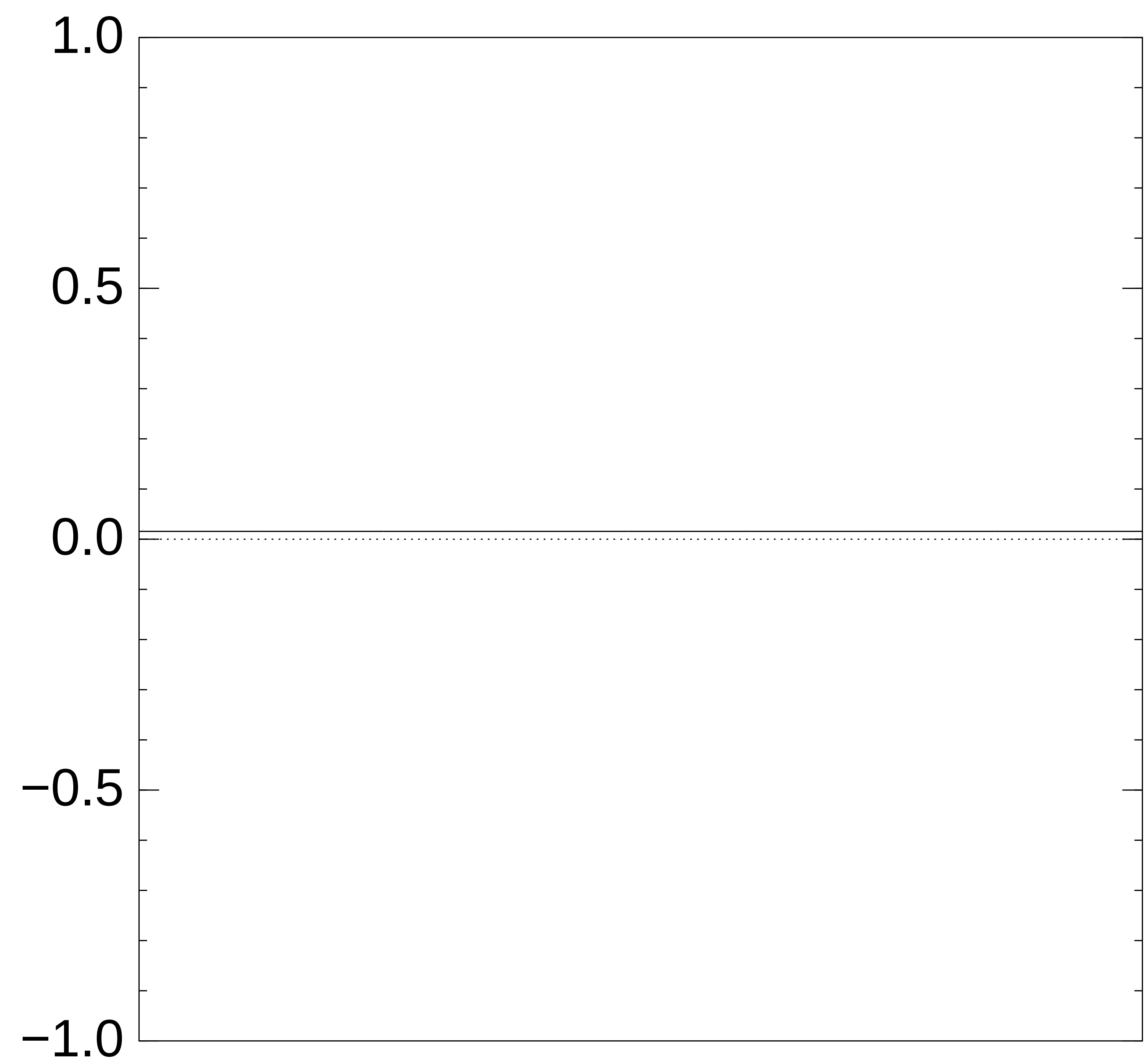
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

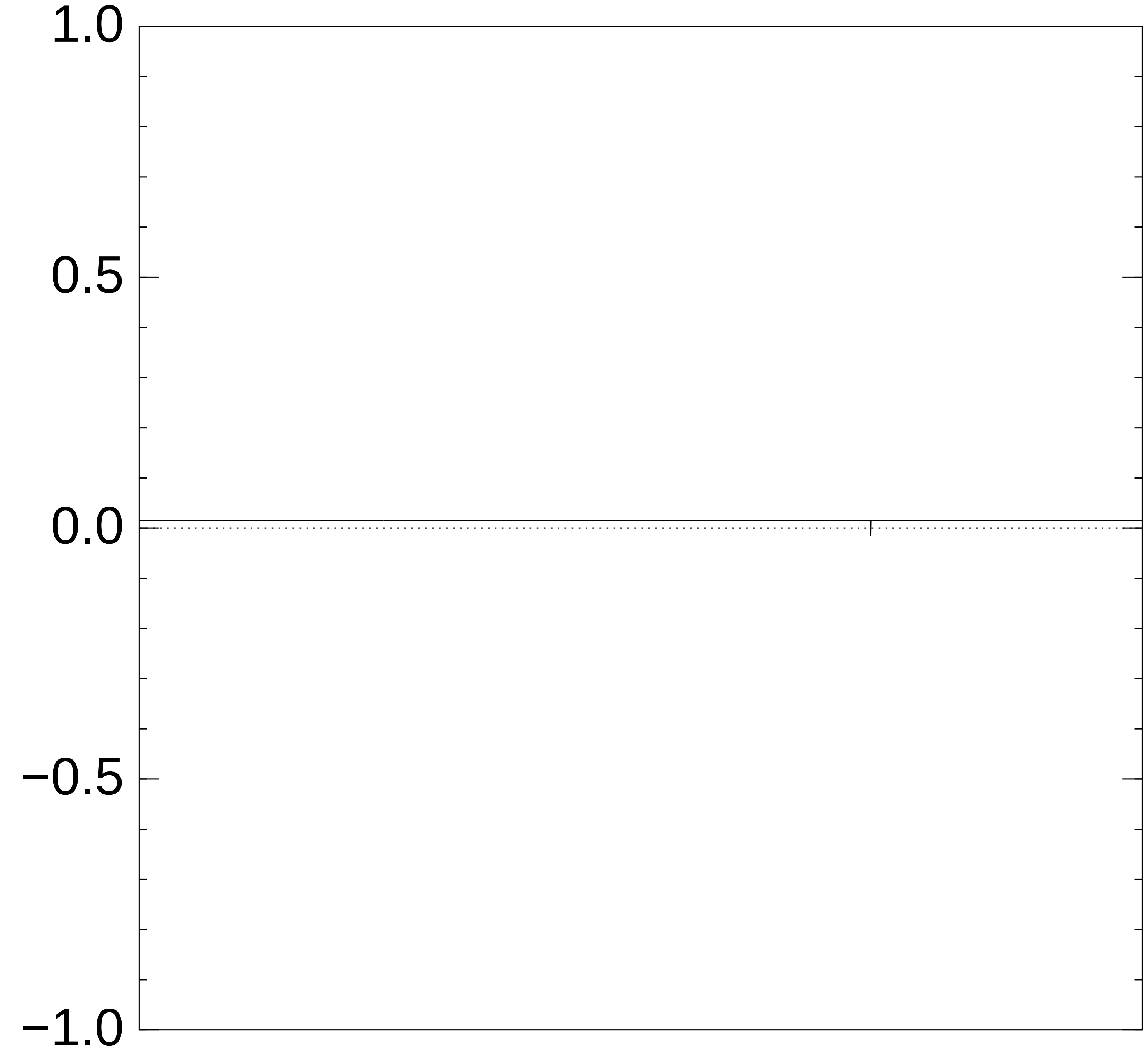
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after Step 1:



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

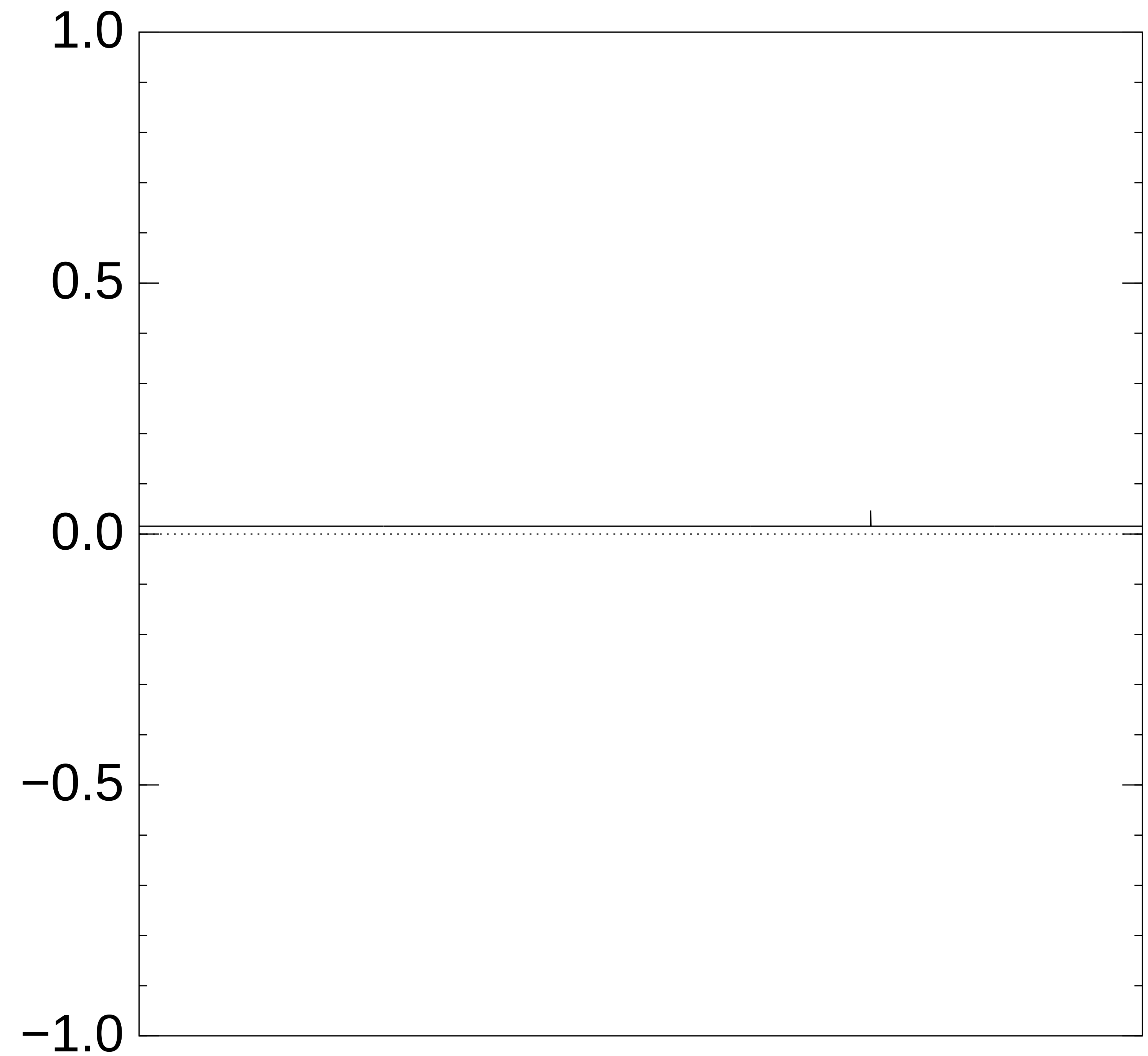
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

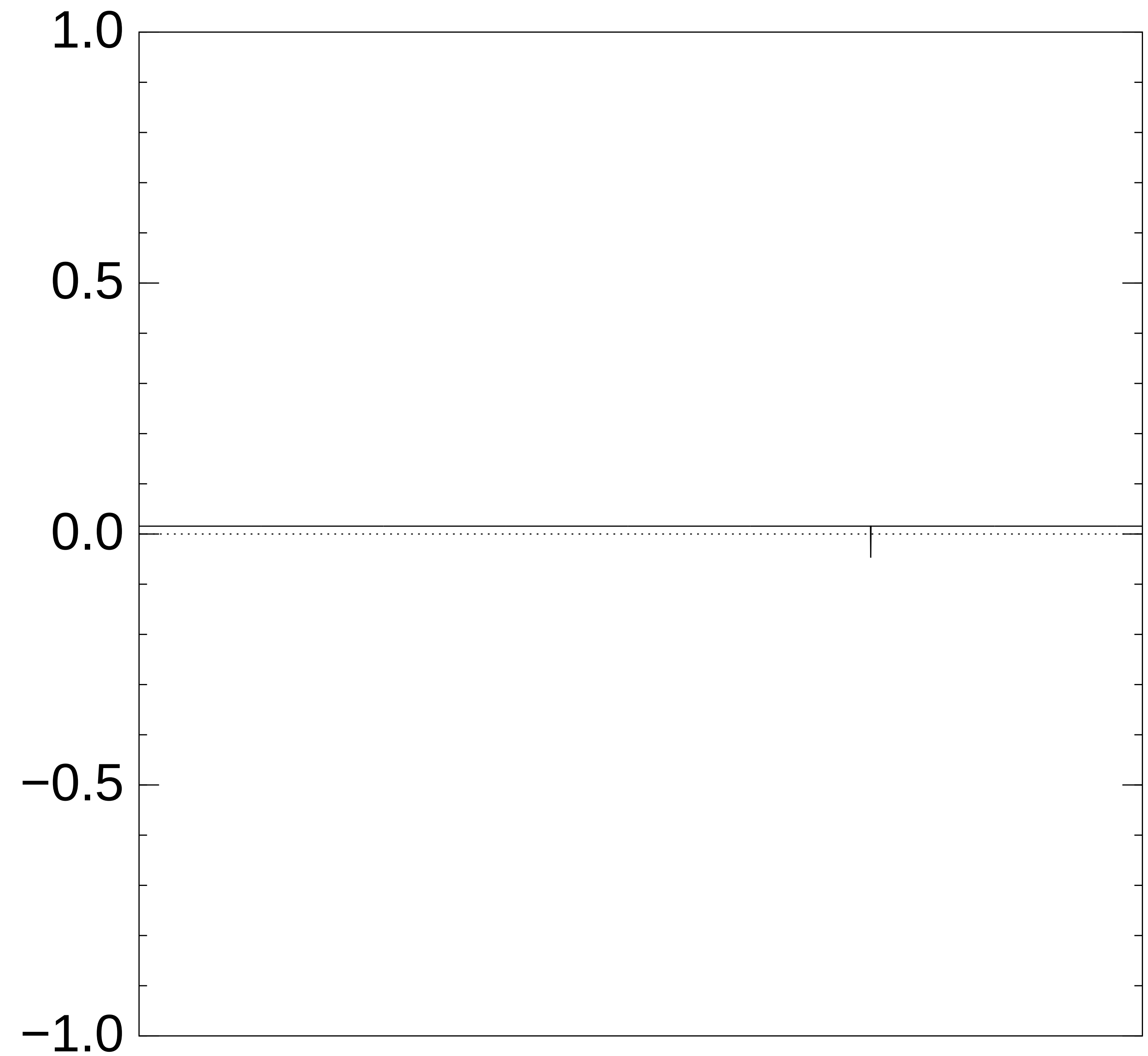
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

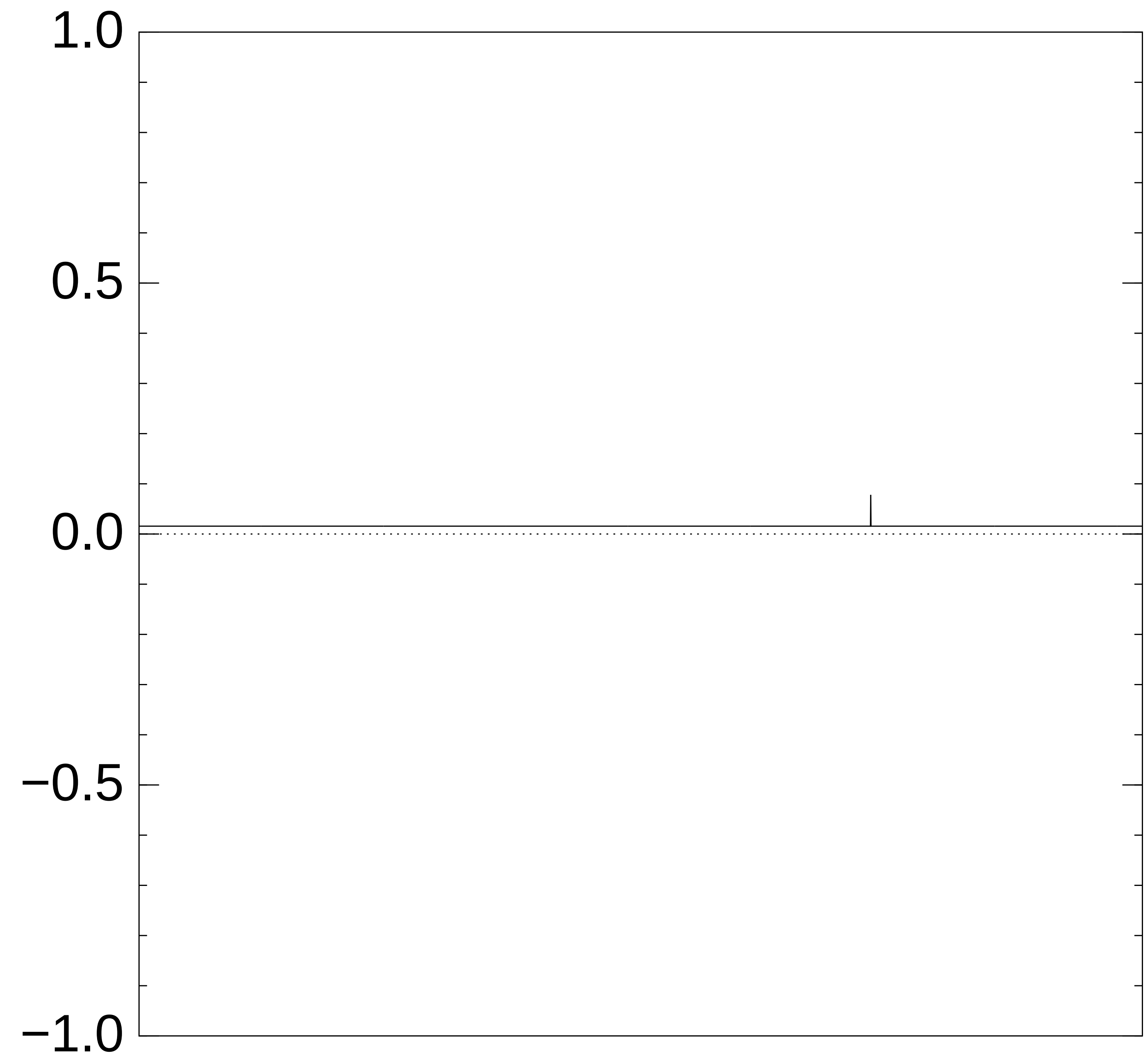
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

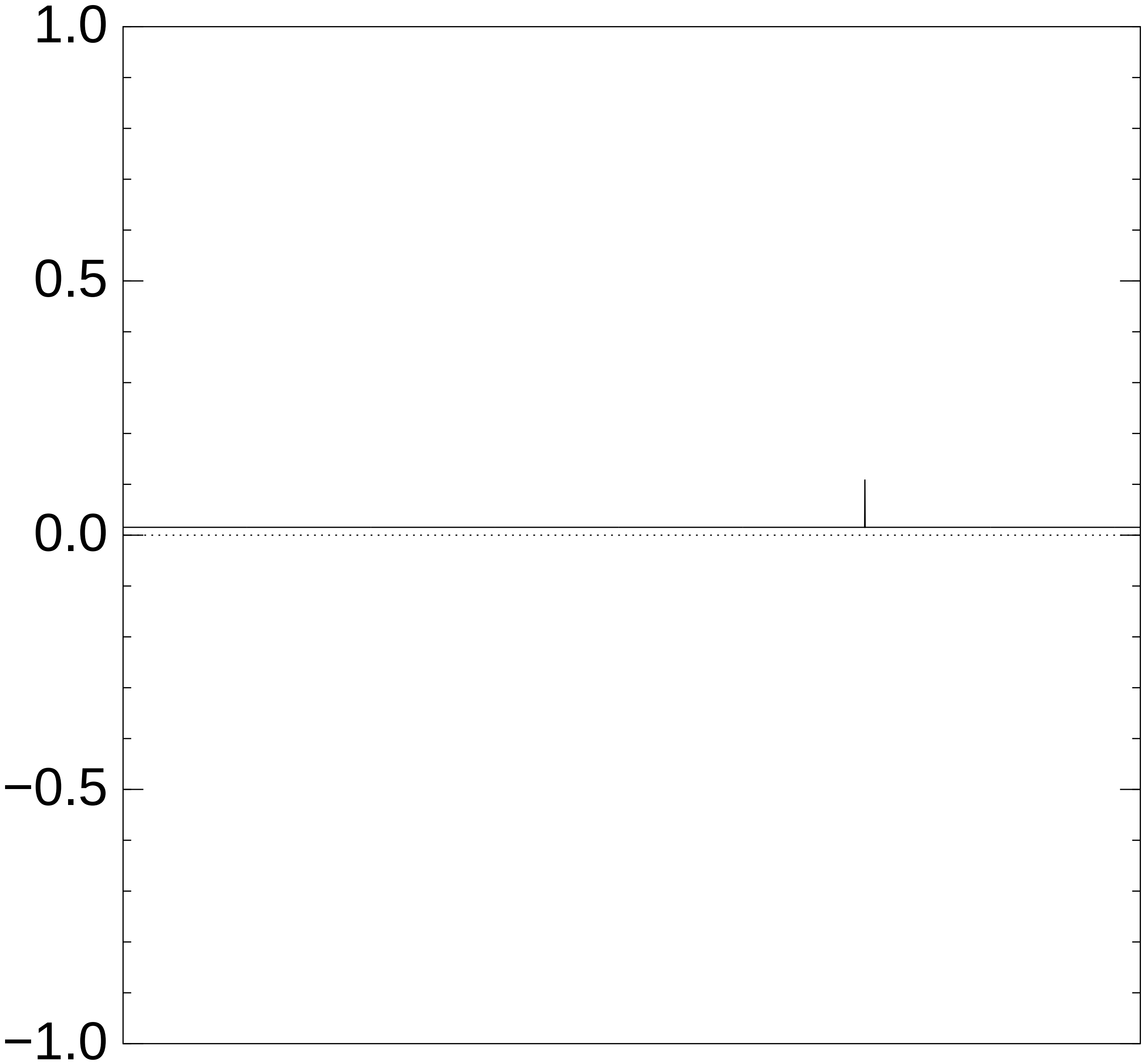
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

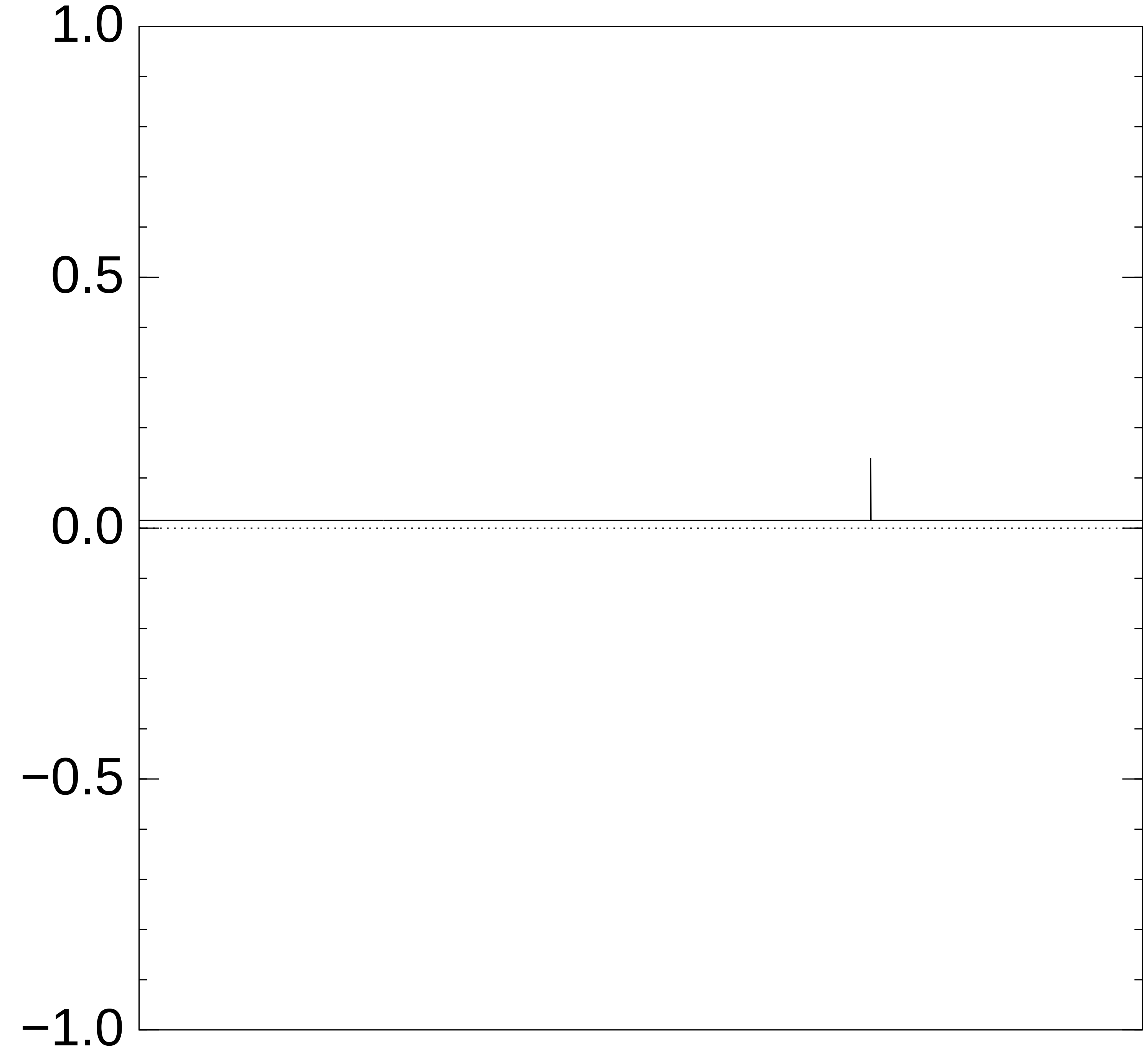
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $4 \times$ (Step 1 + Step 2):



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

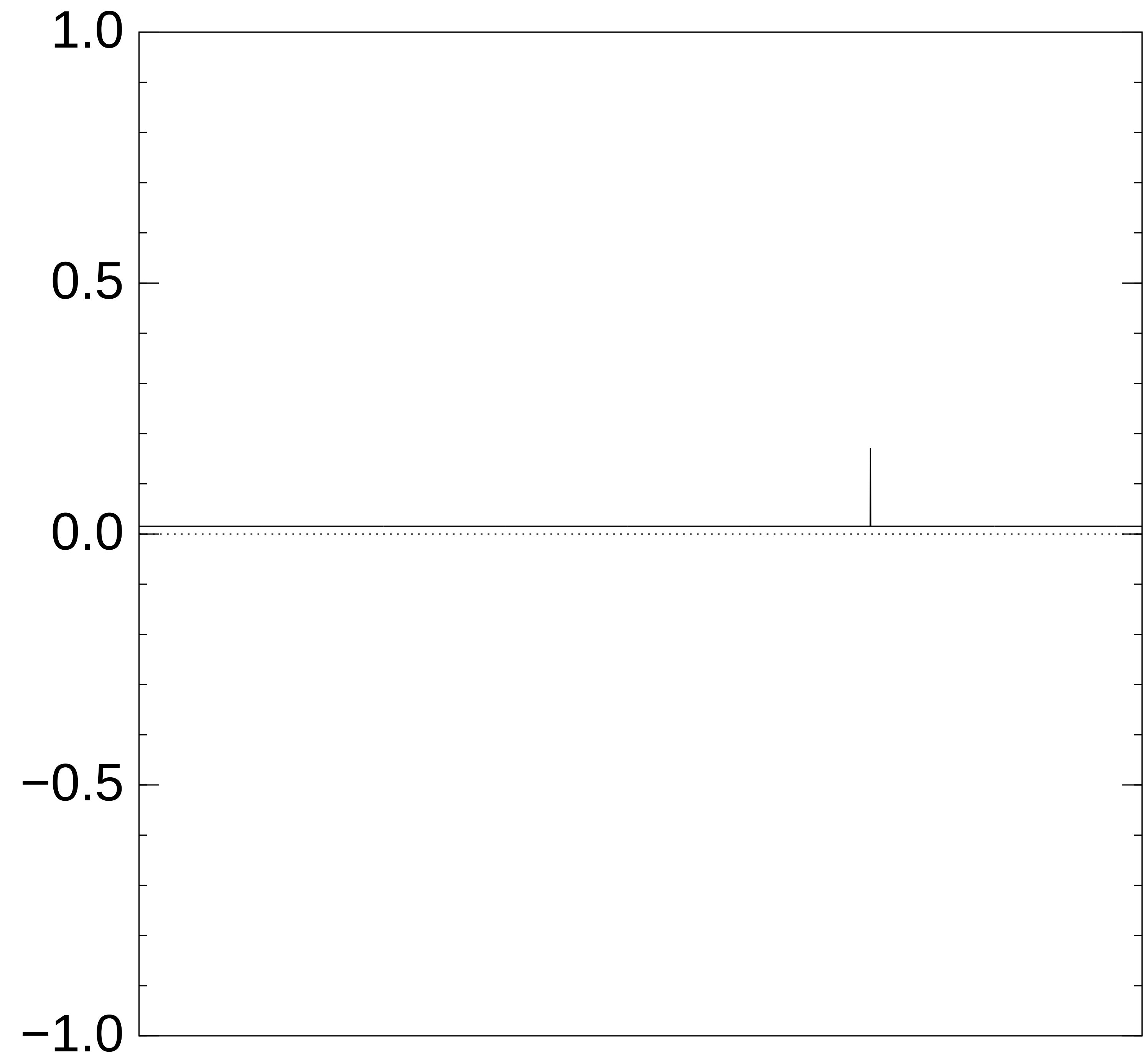
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

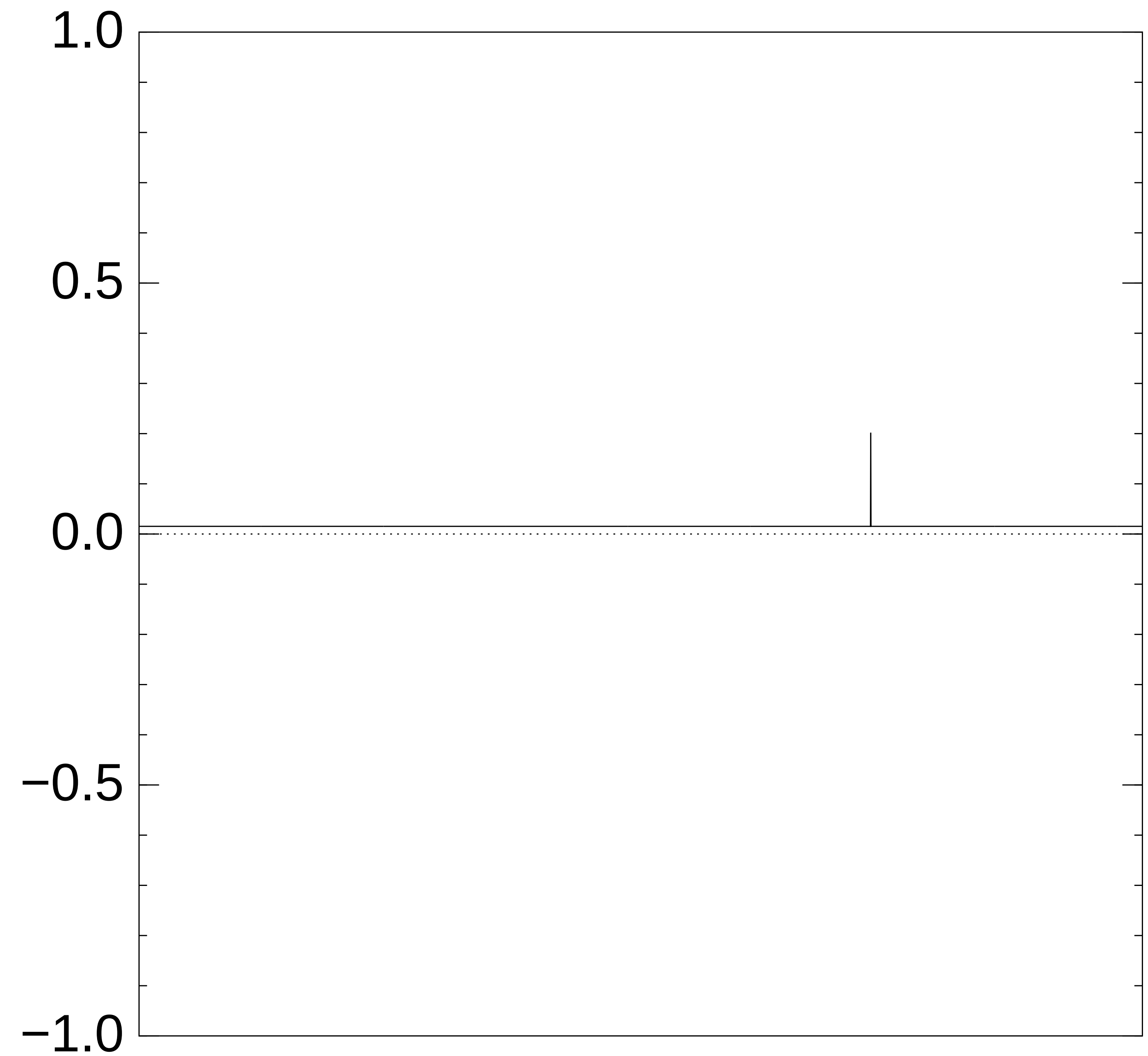
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $6 \times$ (Step 1 + Step 2):



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

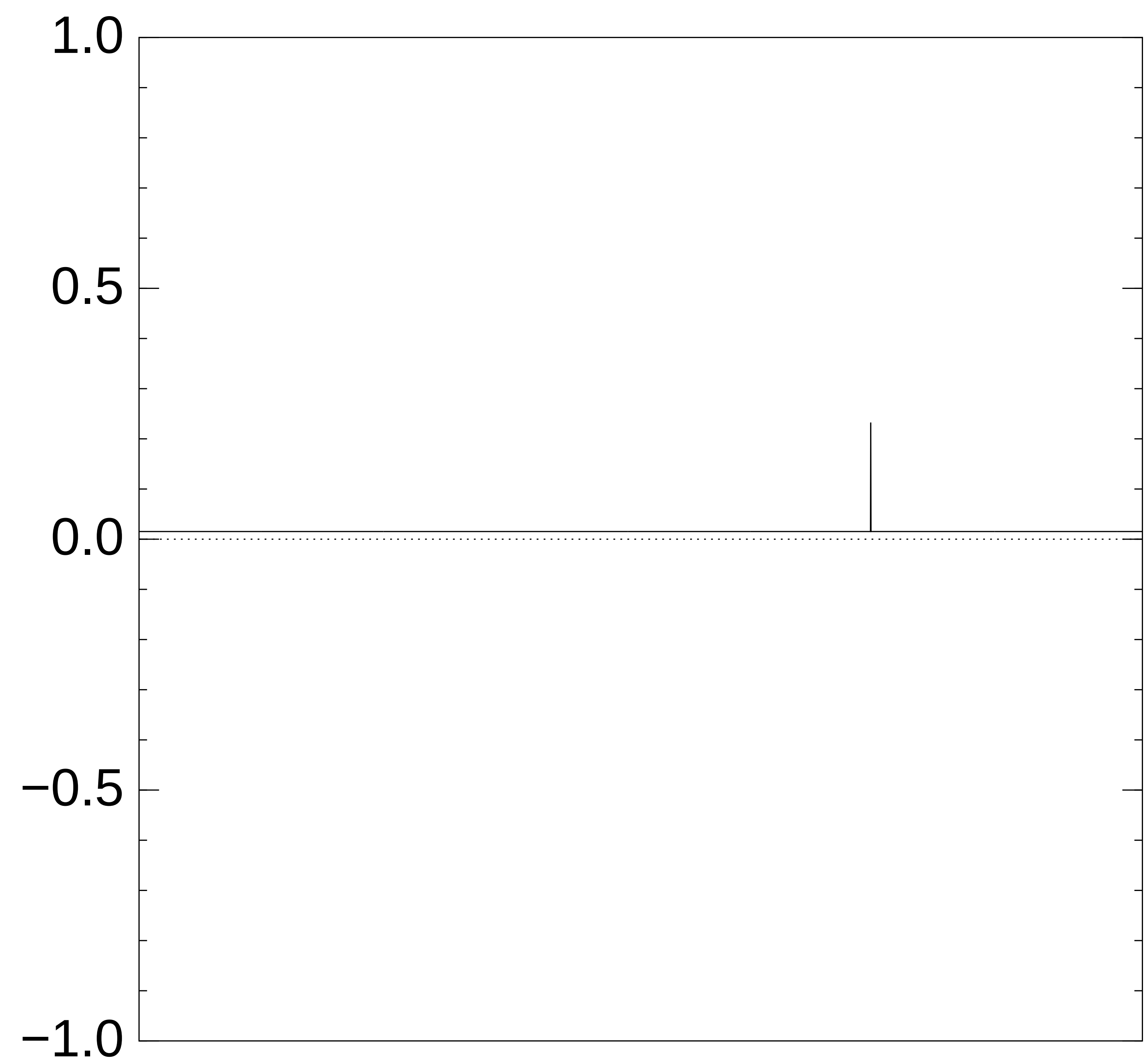
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

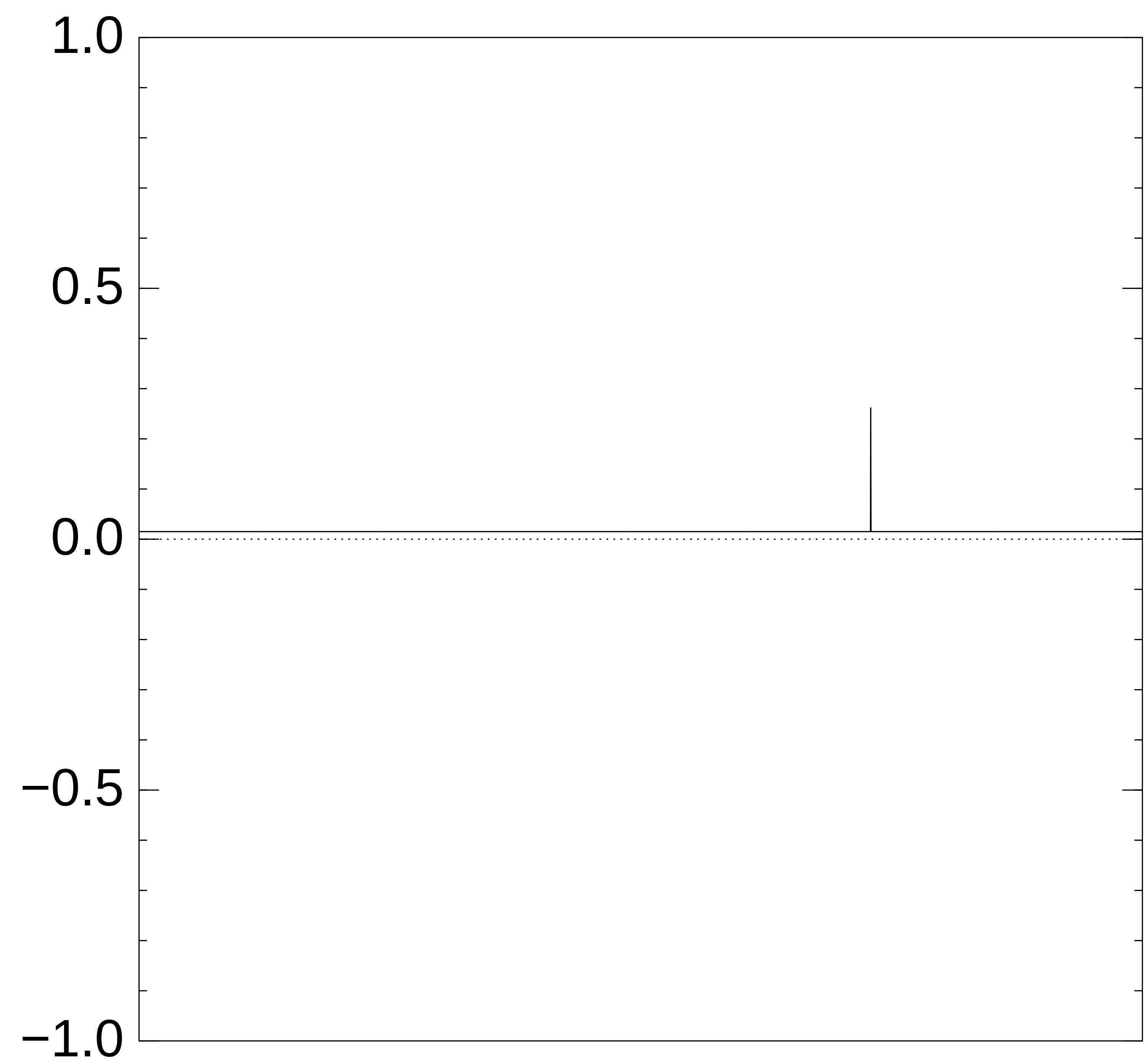
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $8 \times$ (Step 1 + Step 2):



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

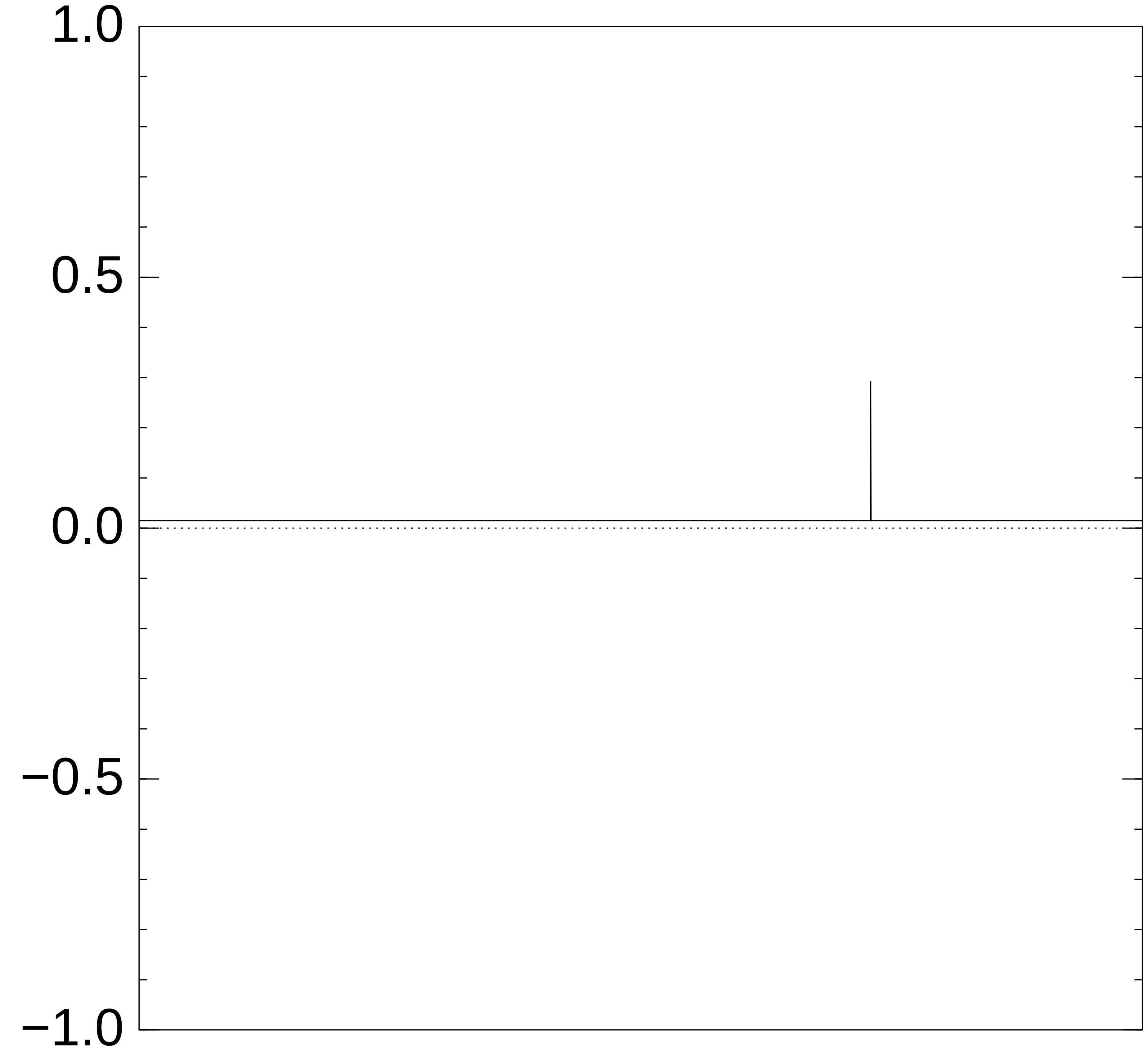
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

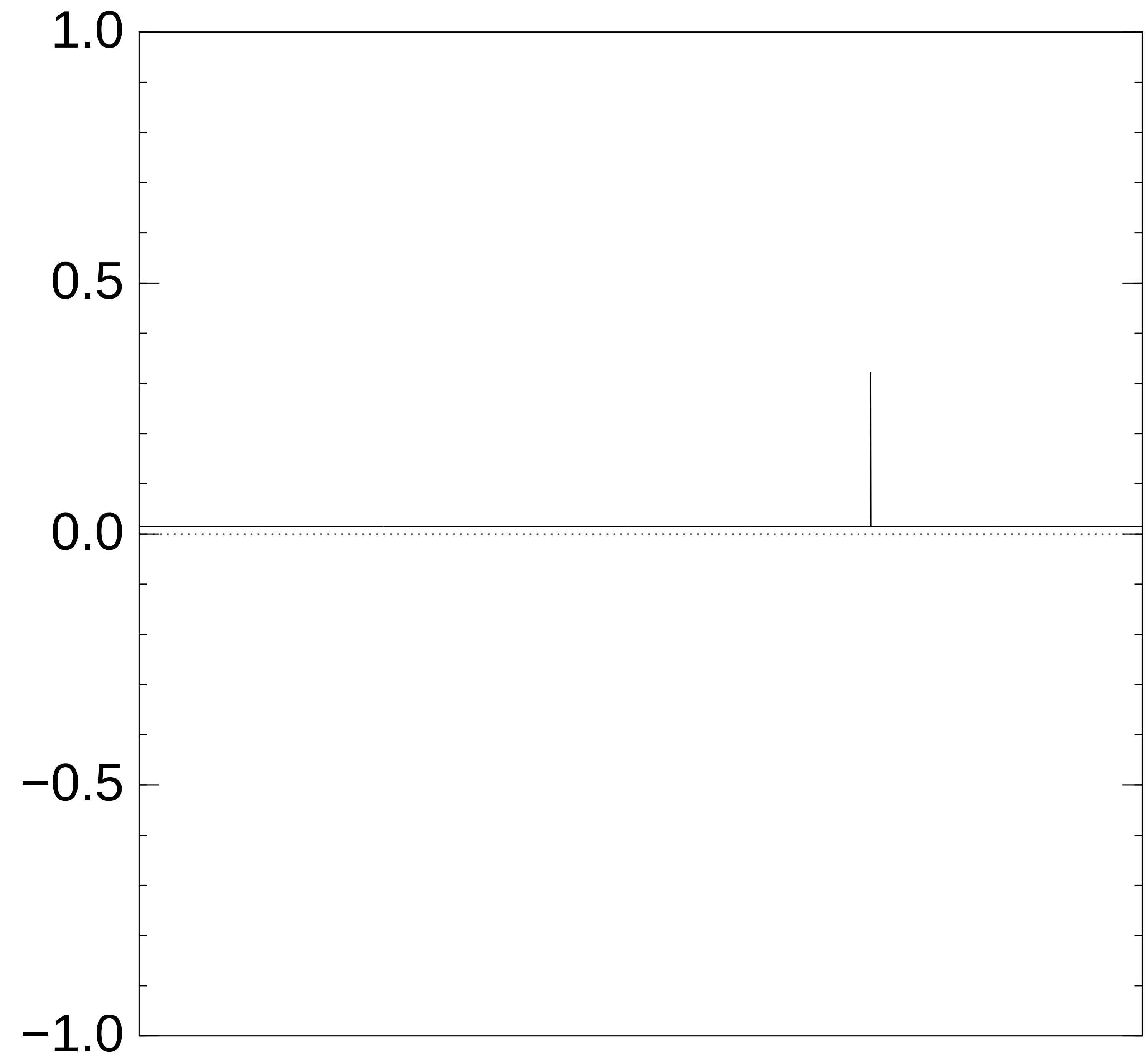
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

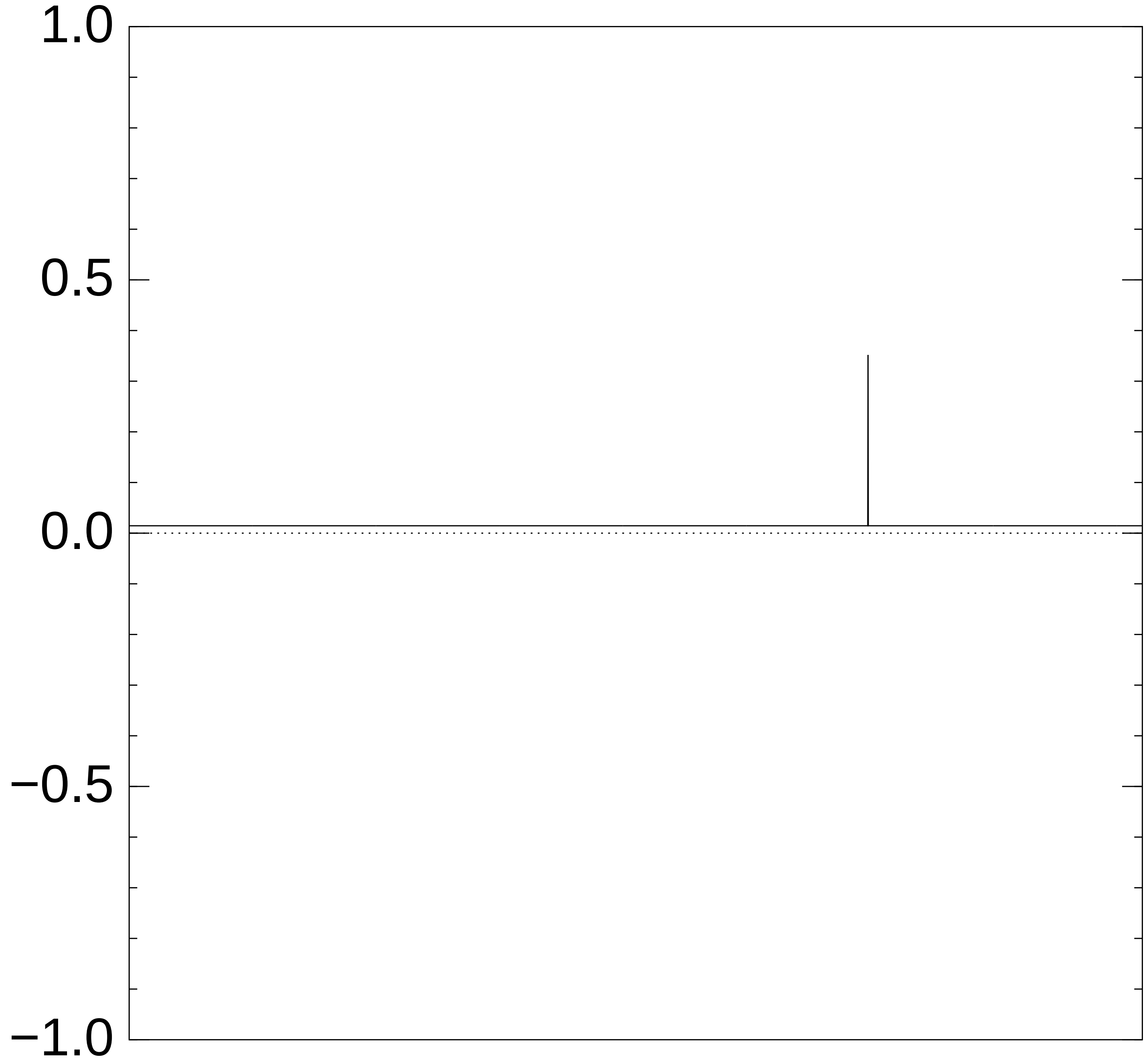
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $11 \times$ (Step 1 + Step 2):



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

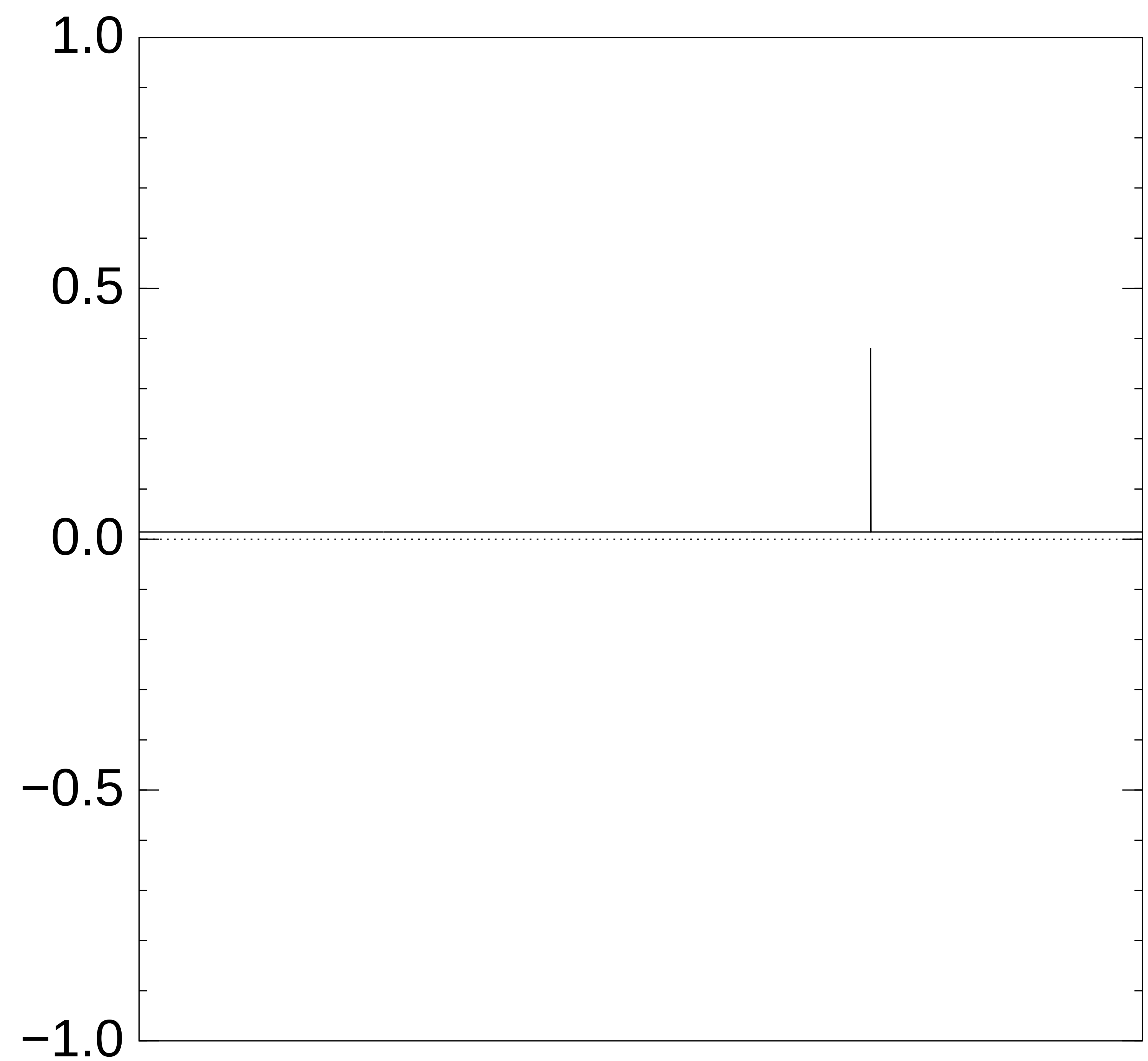
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

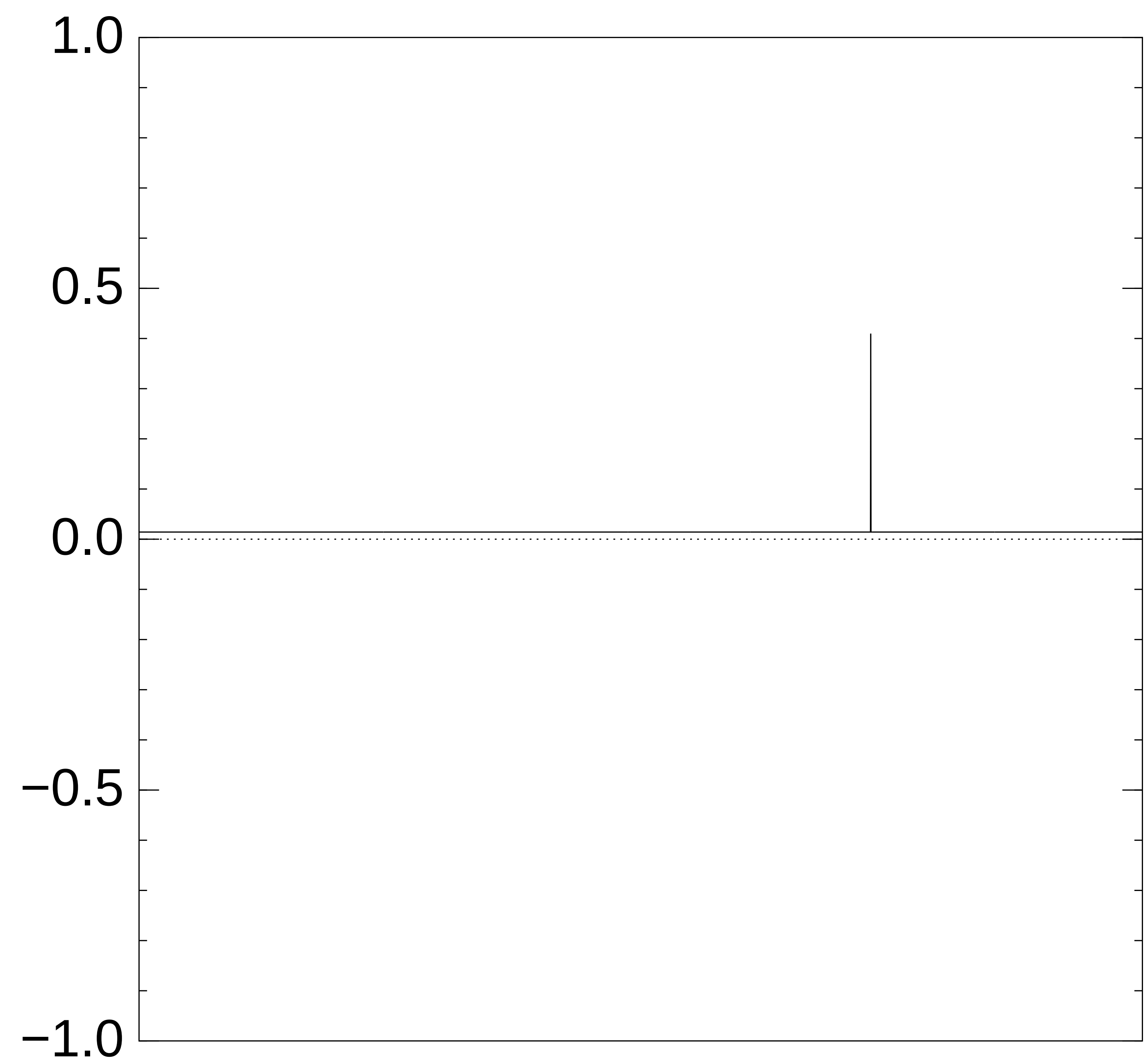
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

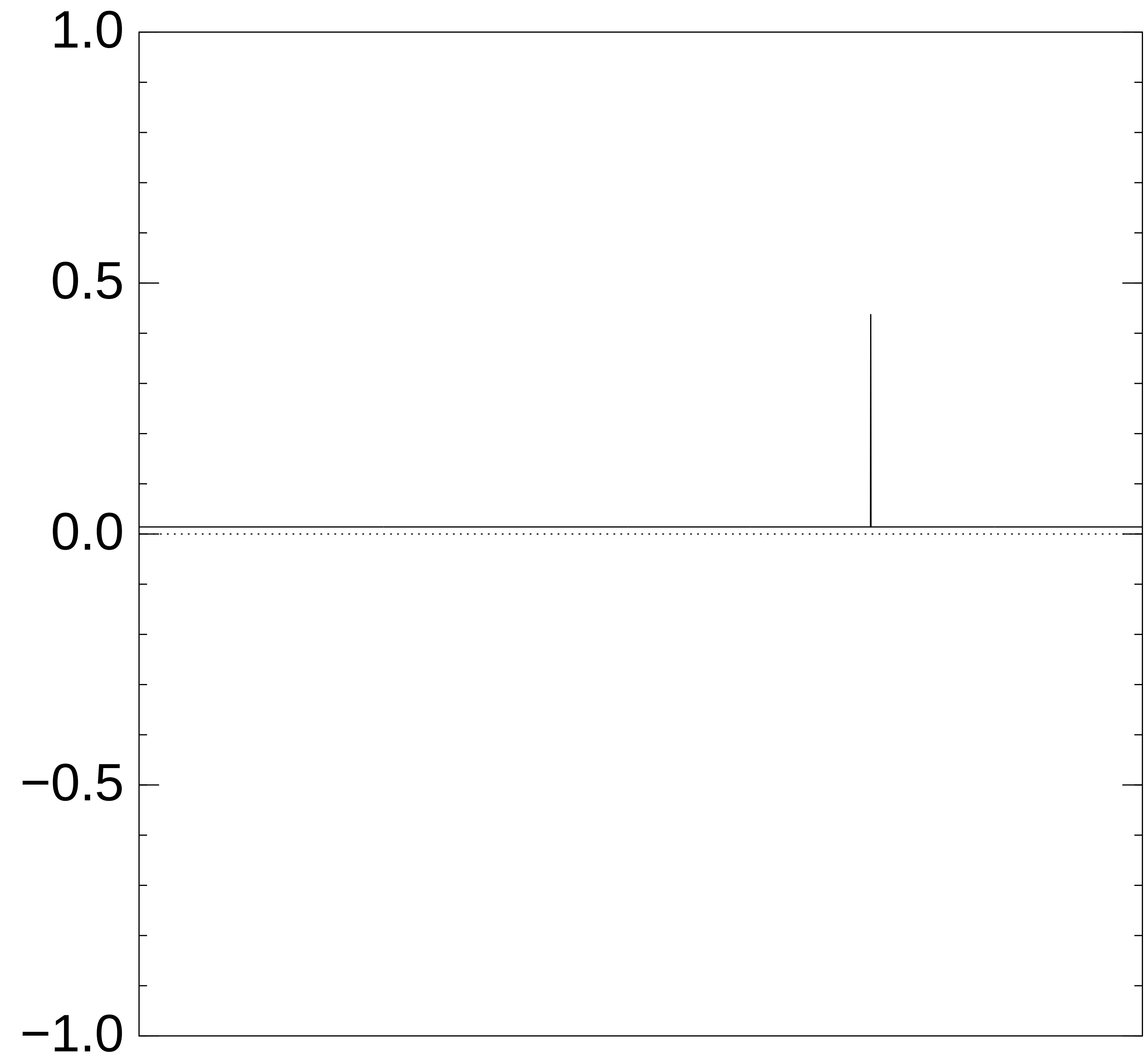
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $14 \times$ (Step 1 + Step 2):



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

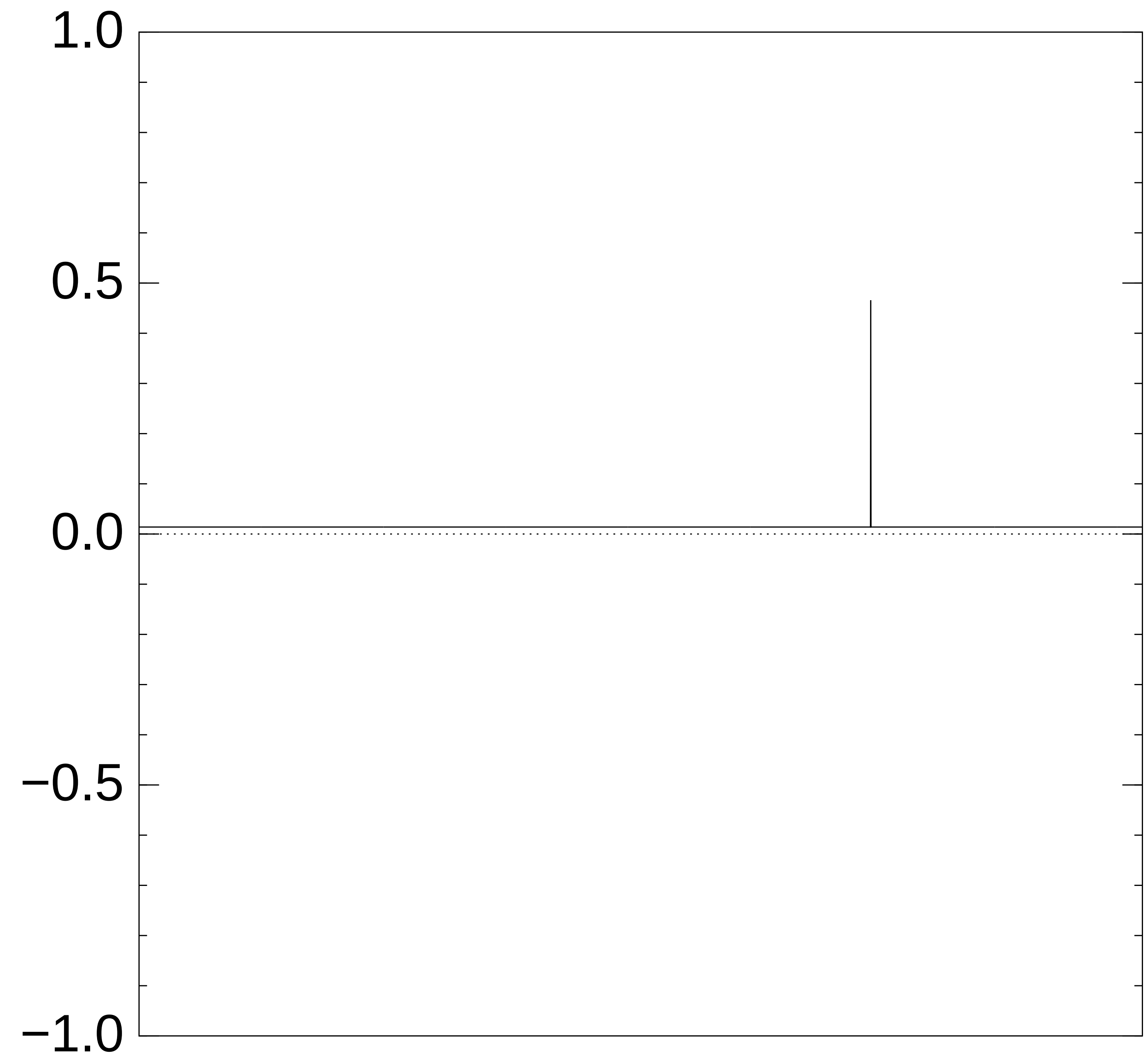
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $15 \times$ (Step 1 + Step 2):



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

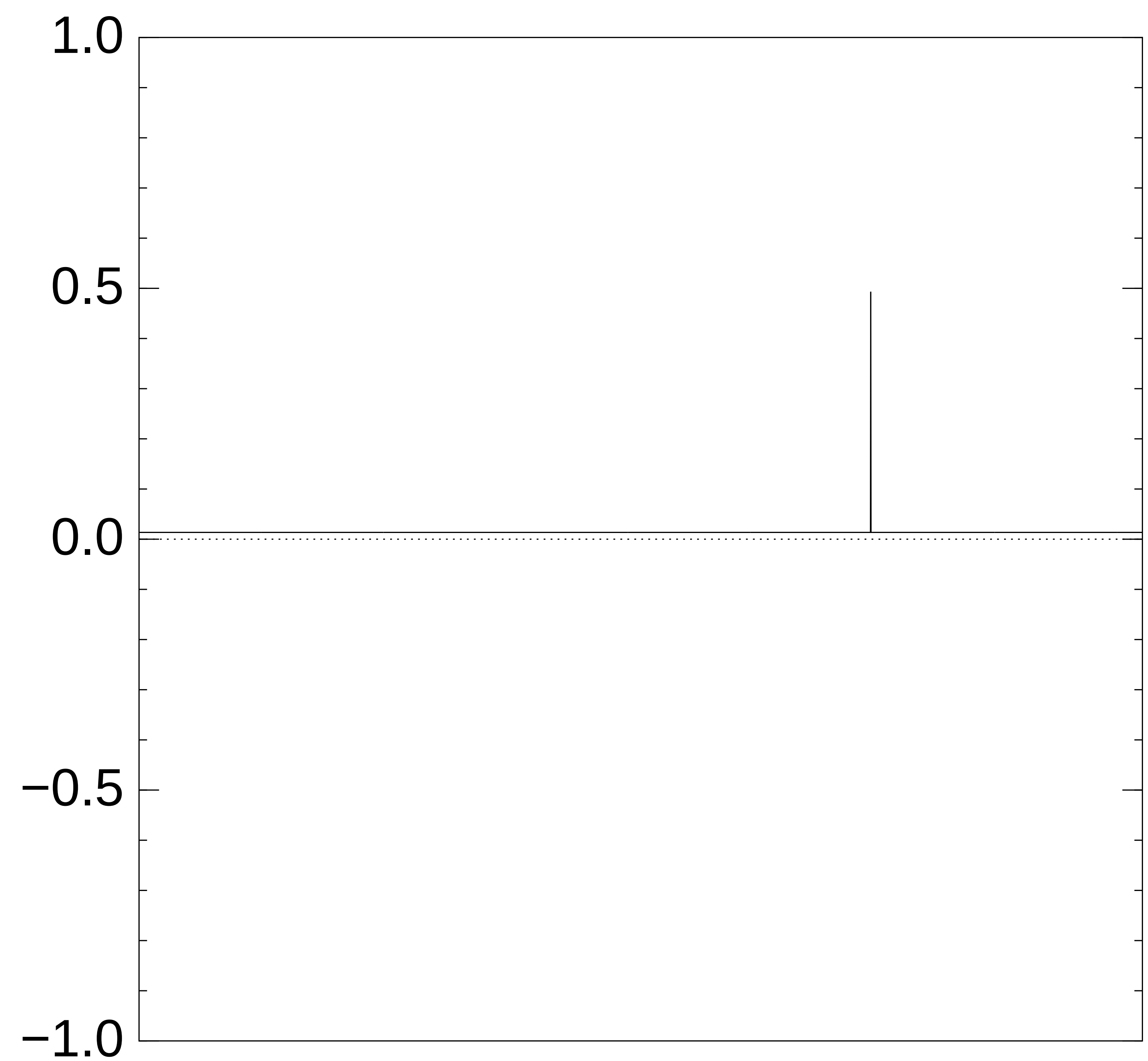
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

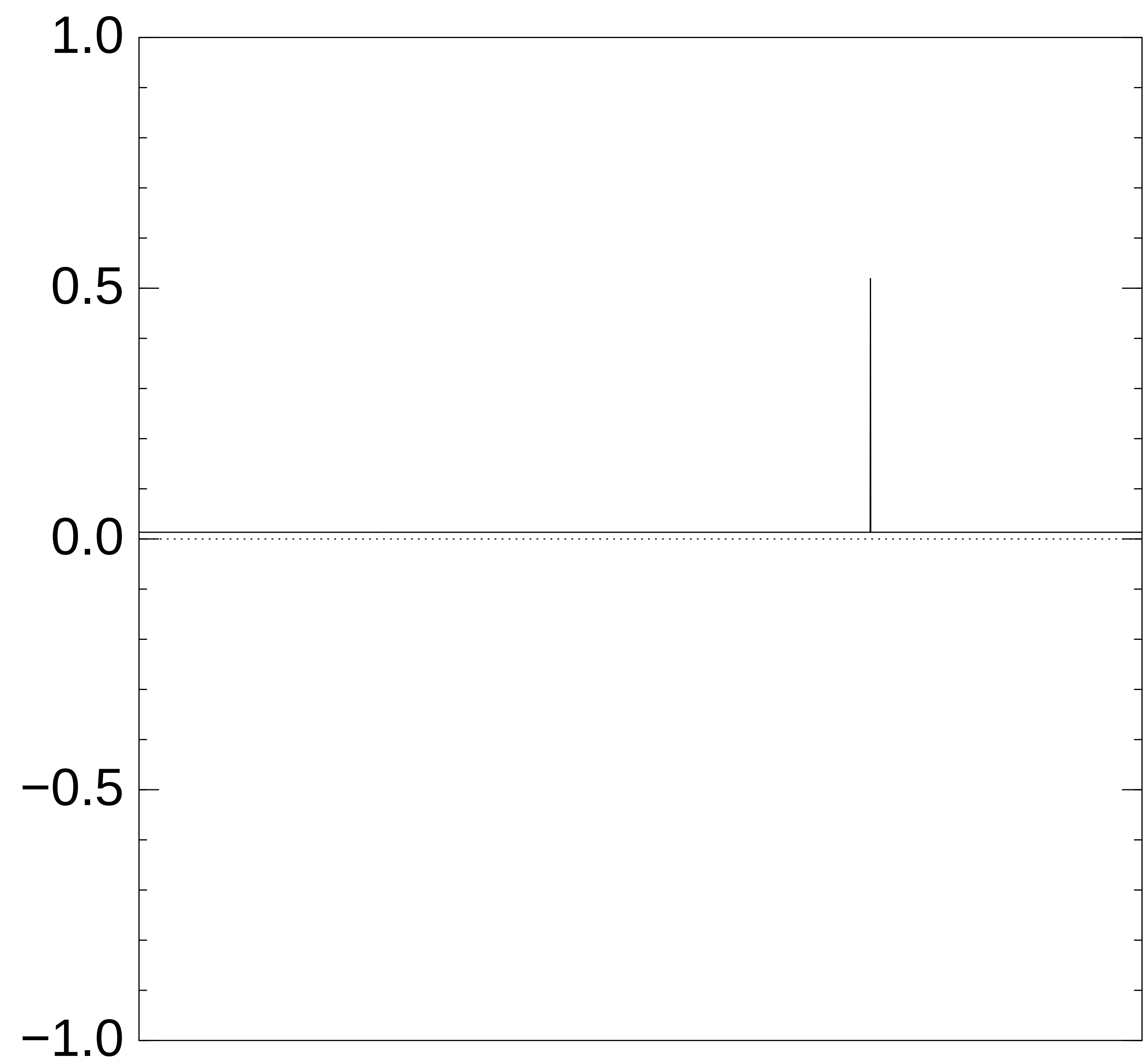
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

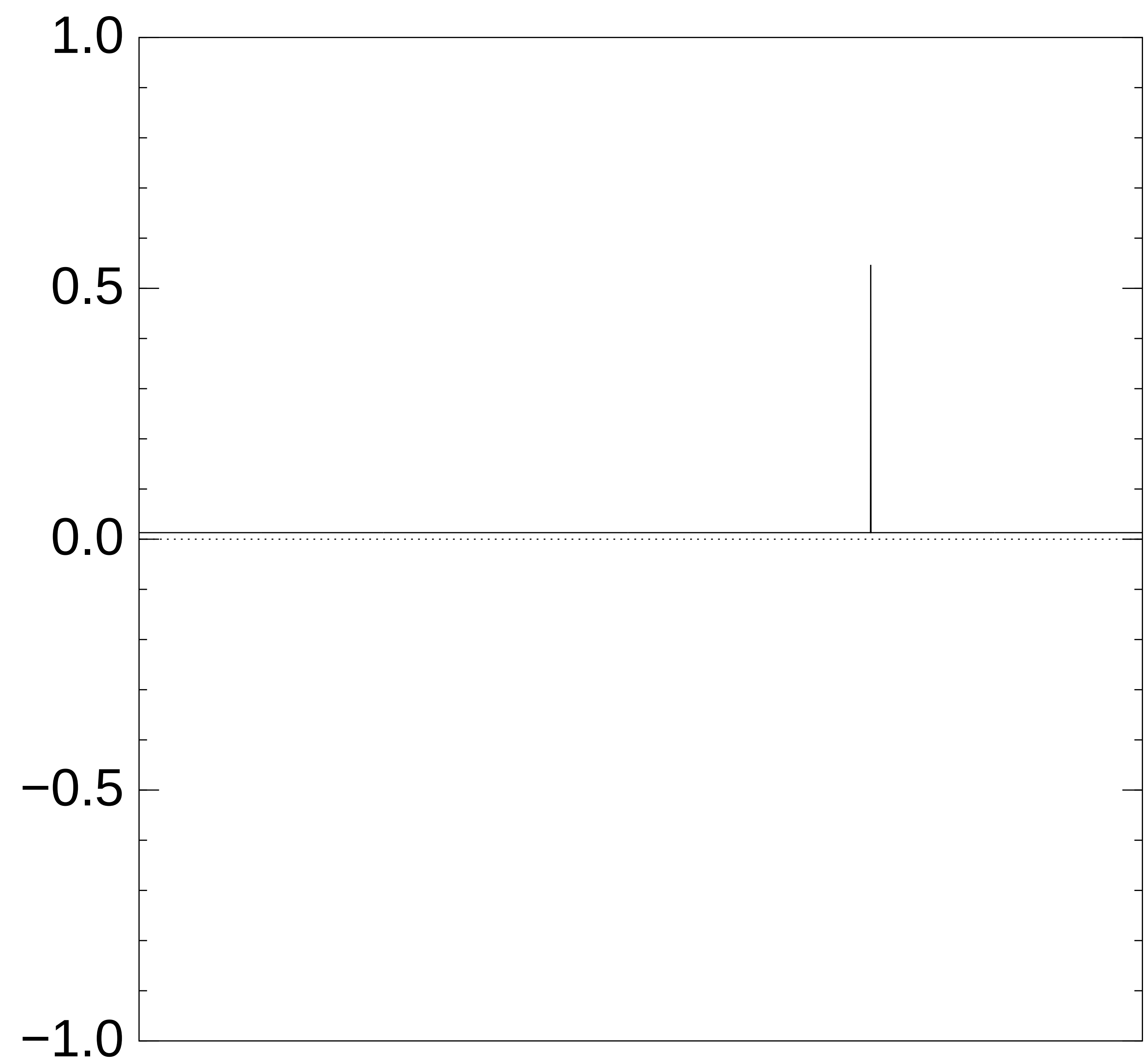
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

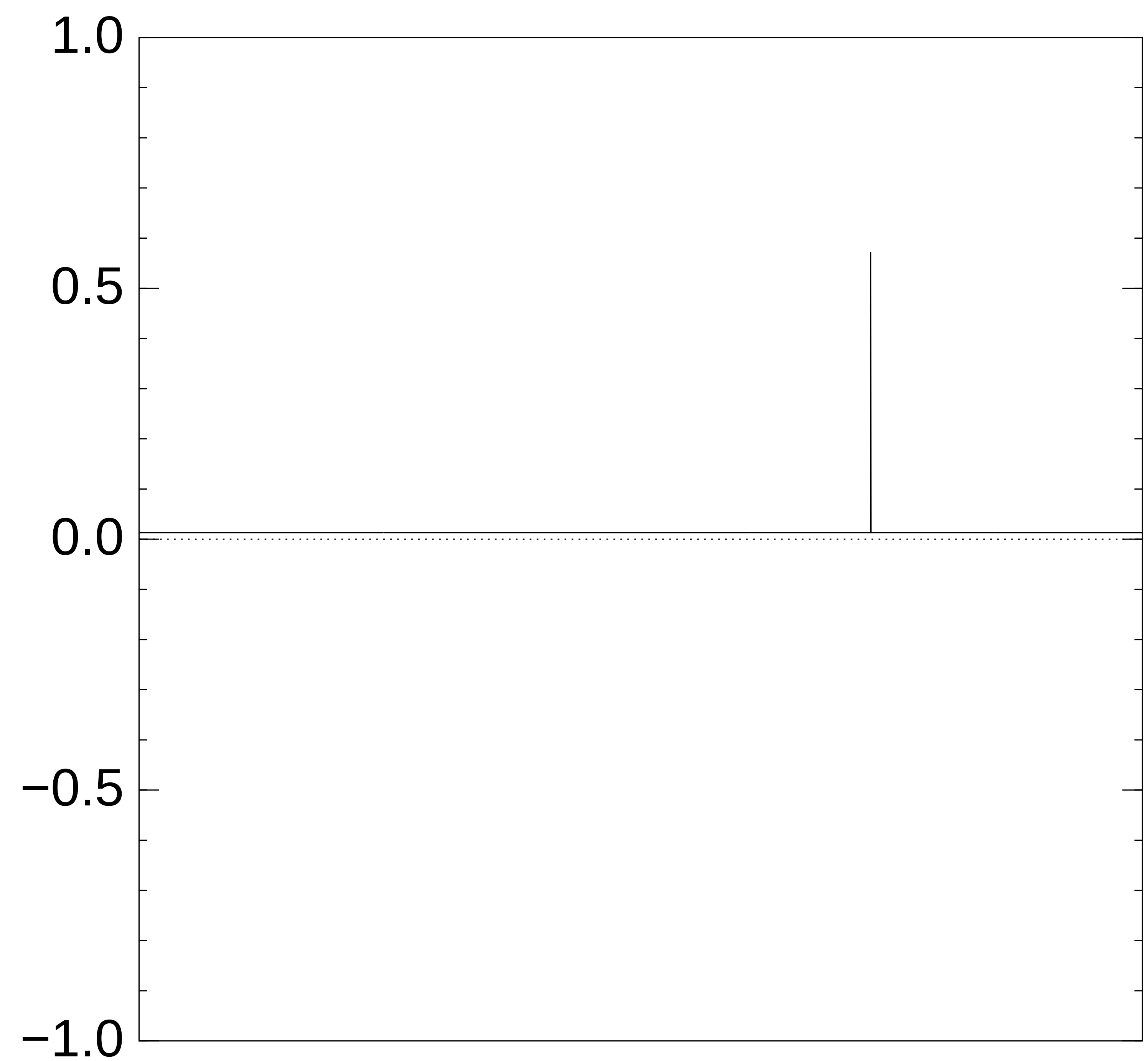
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

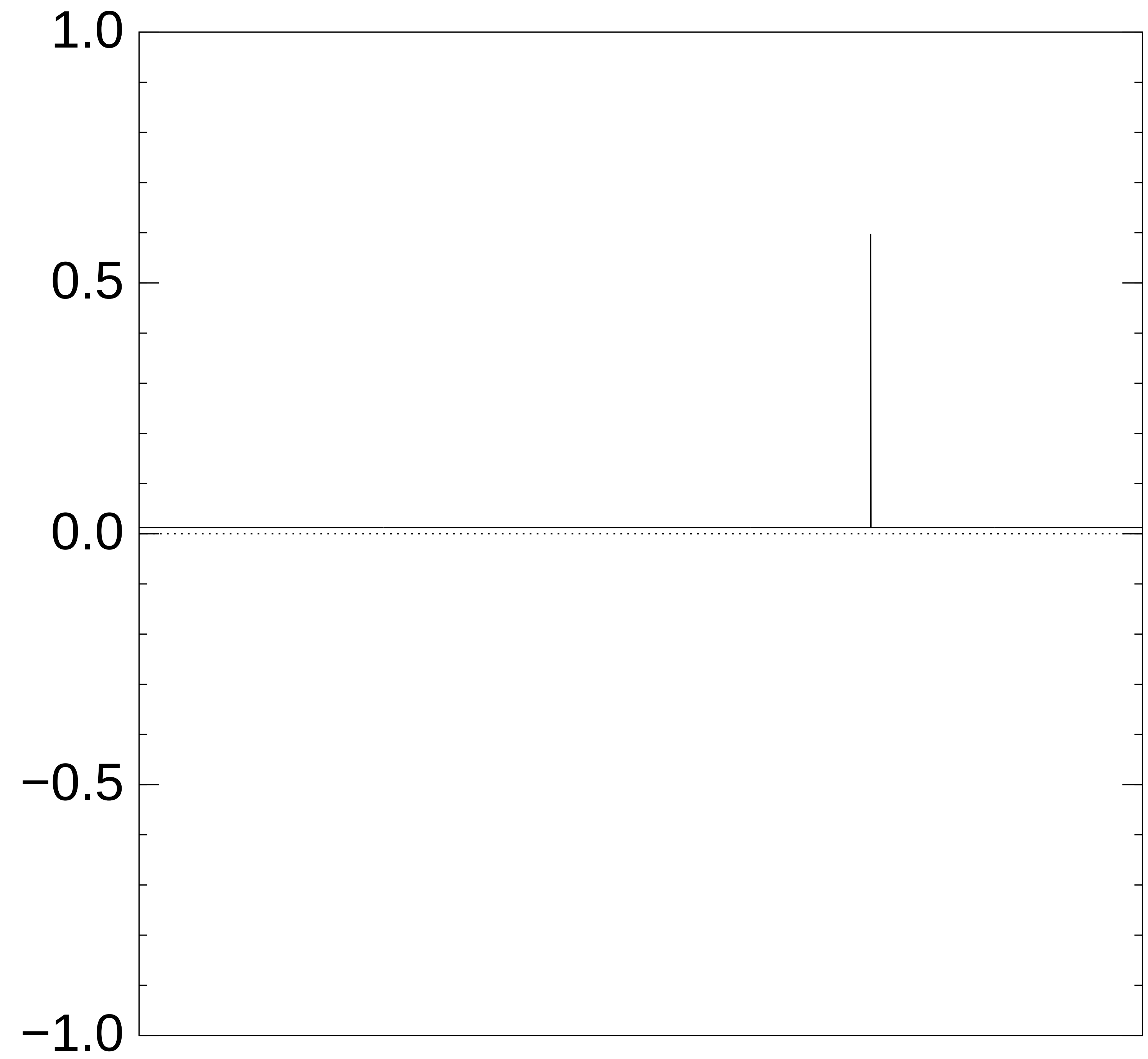
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $20 \times$ (Step 1 + Step 2):



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

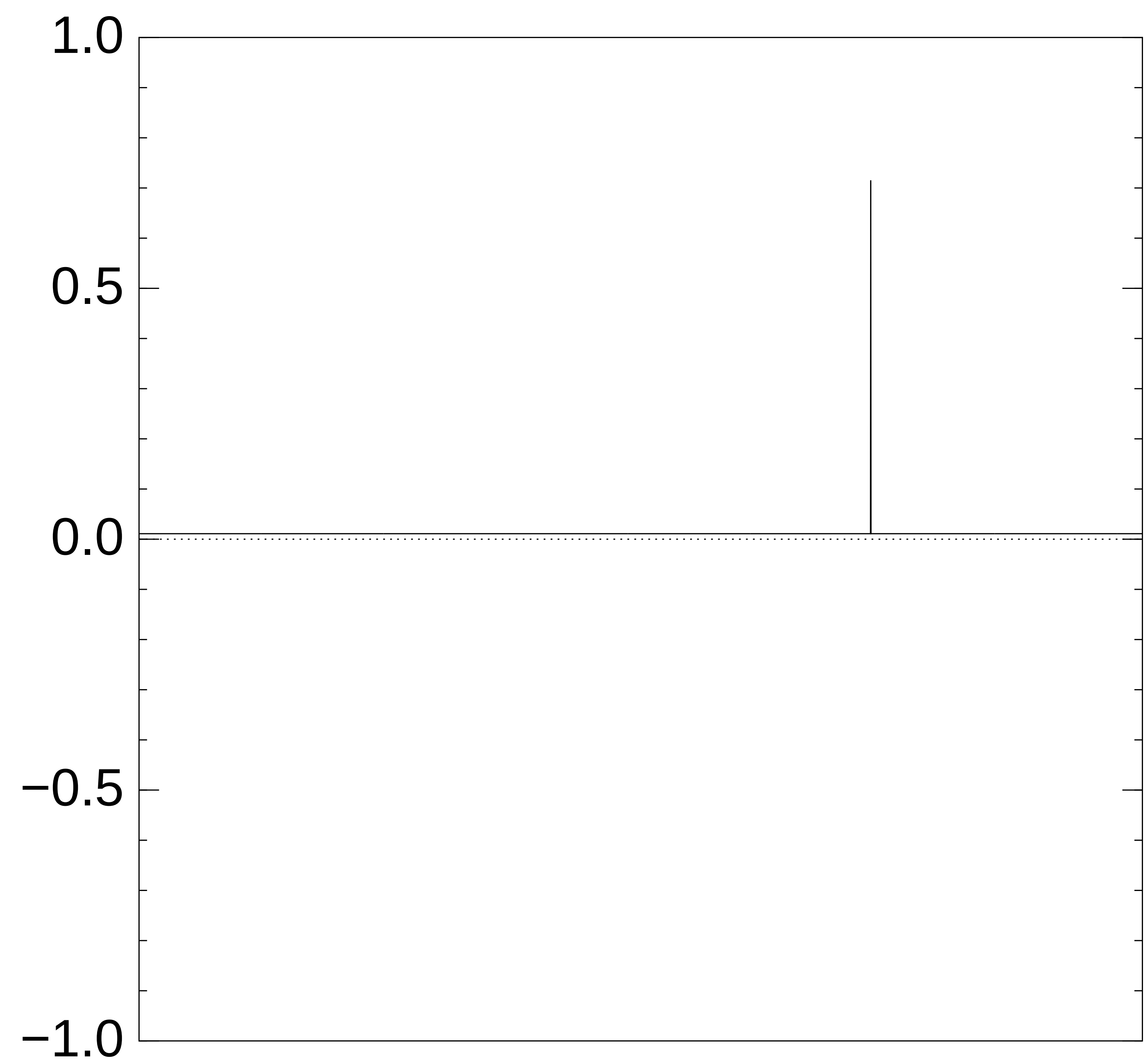
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

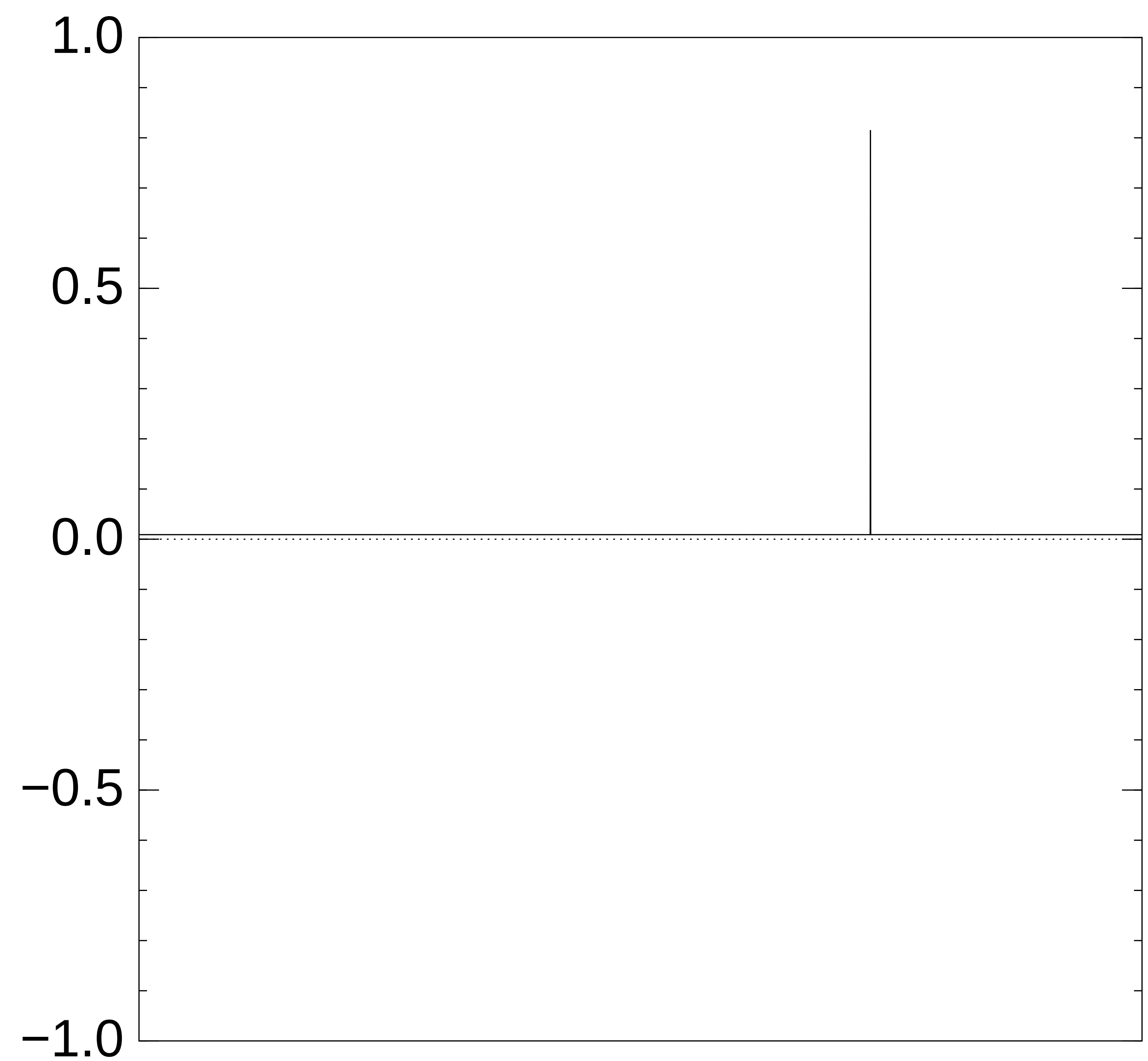
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $30 \times$ (Step 1 + Step 2):



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

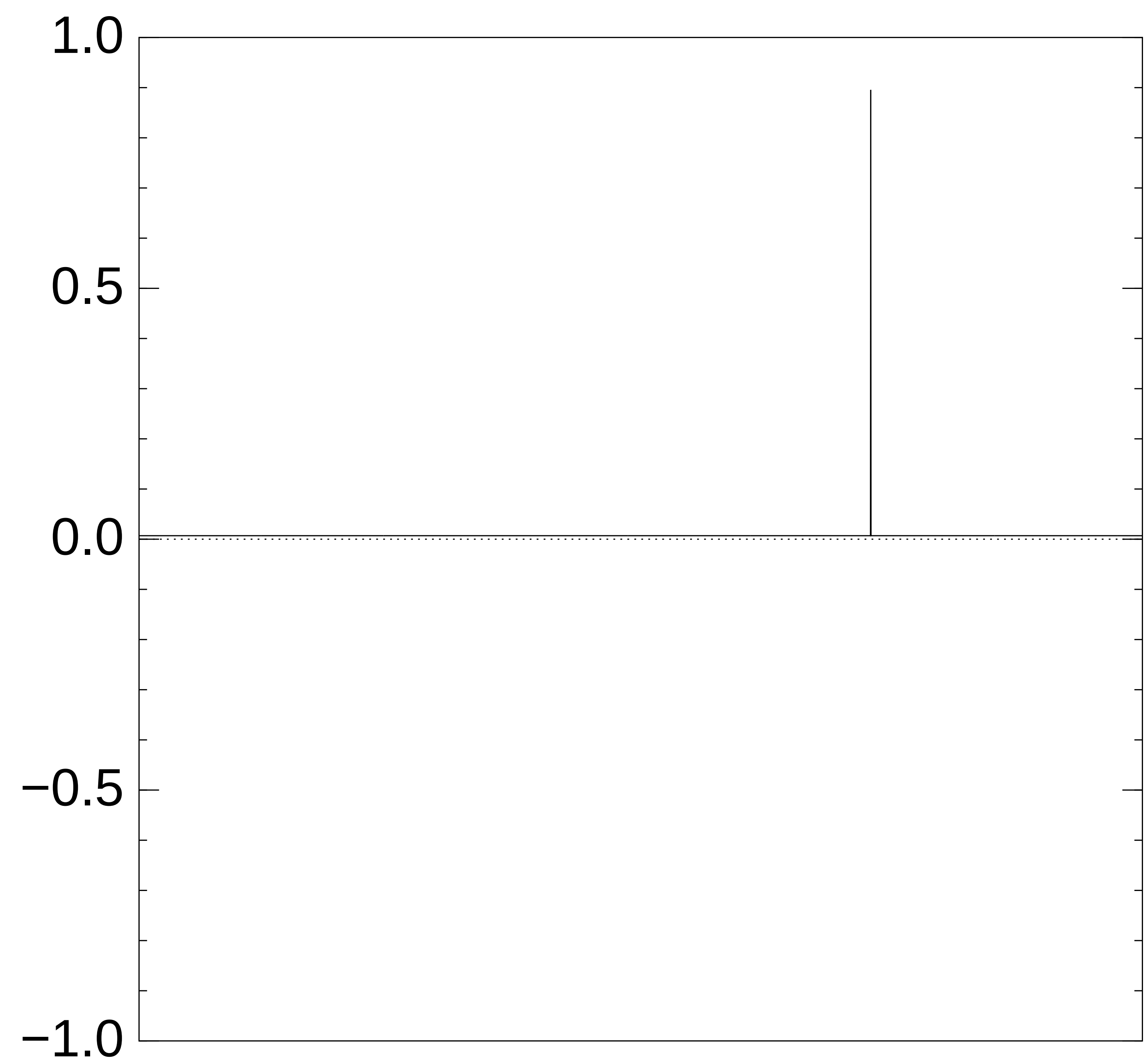
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



Good moment to stop, measure.

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

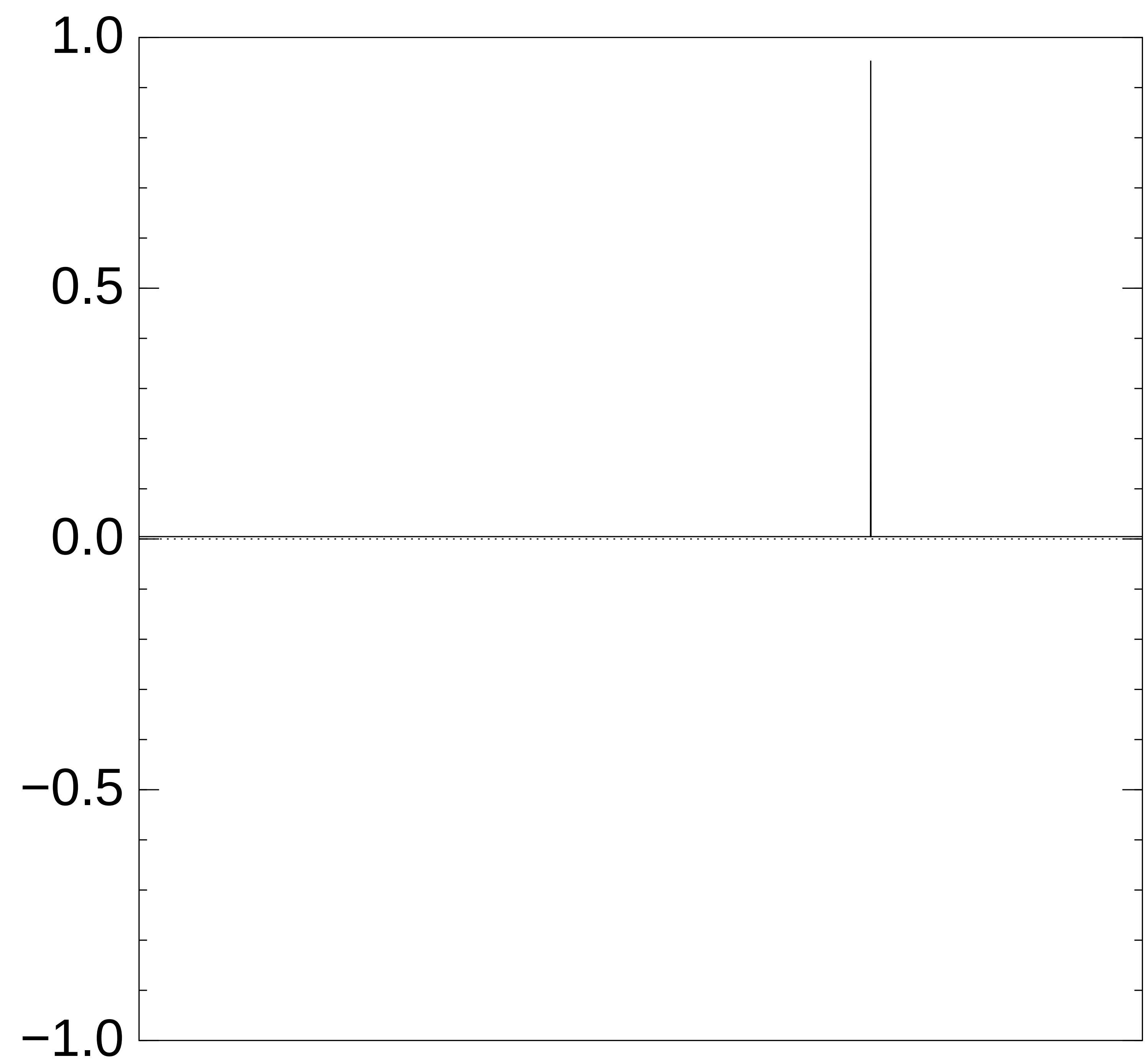
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

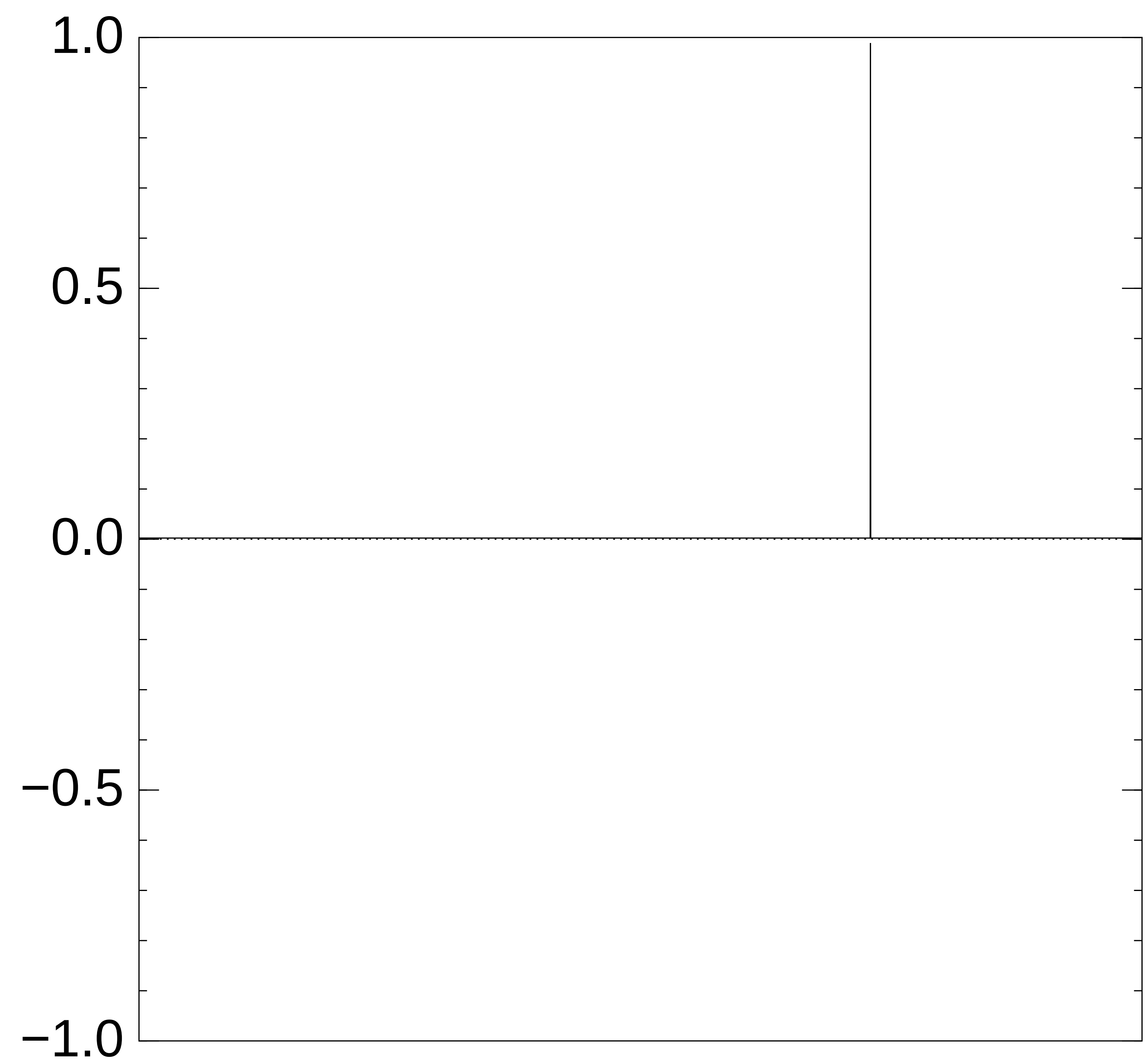
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$ for an example with $n = 12$ after $45 \times$ (Step 1 + Step 2):



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

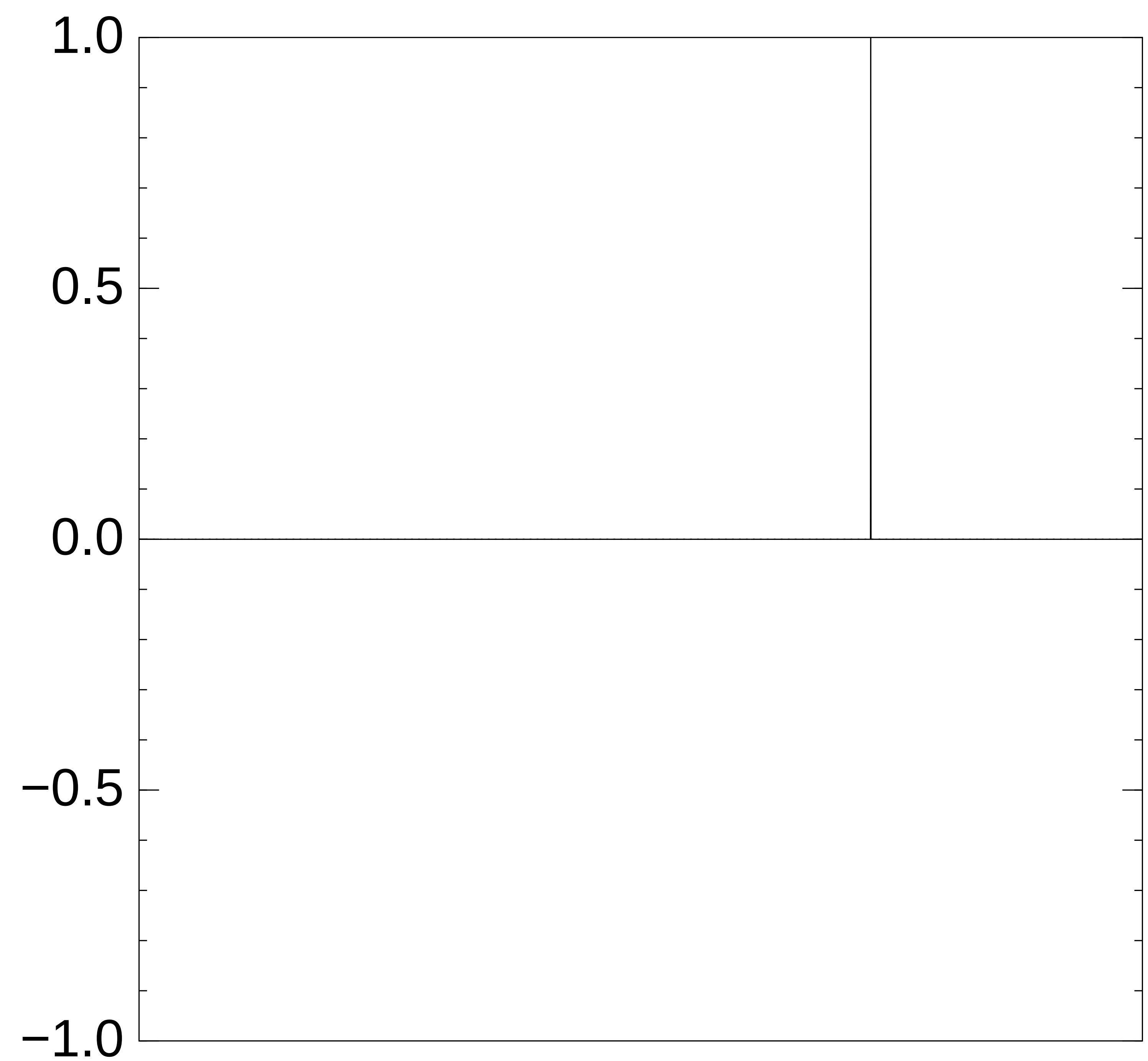
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



Traditional stopping point.

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

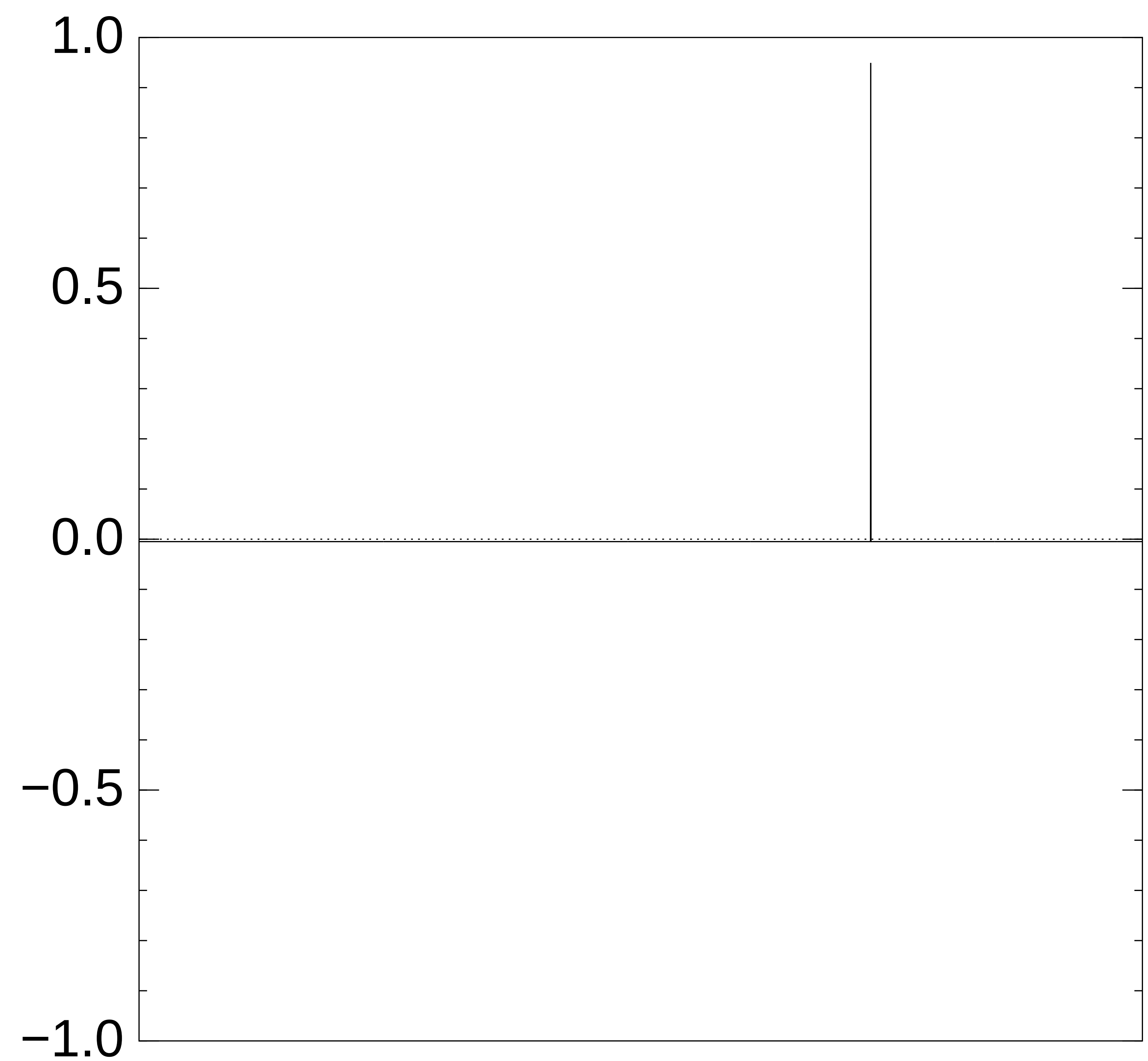
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

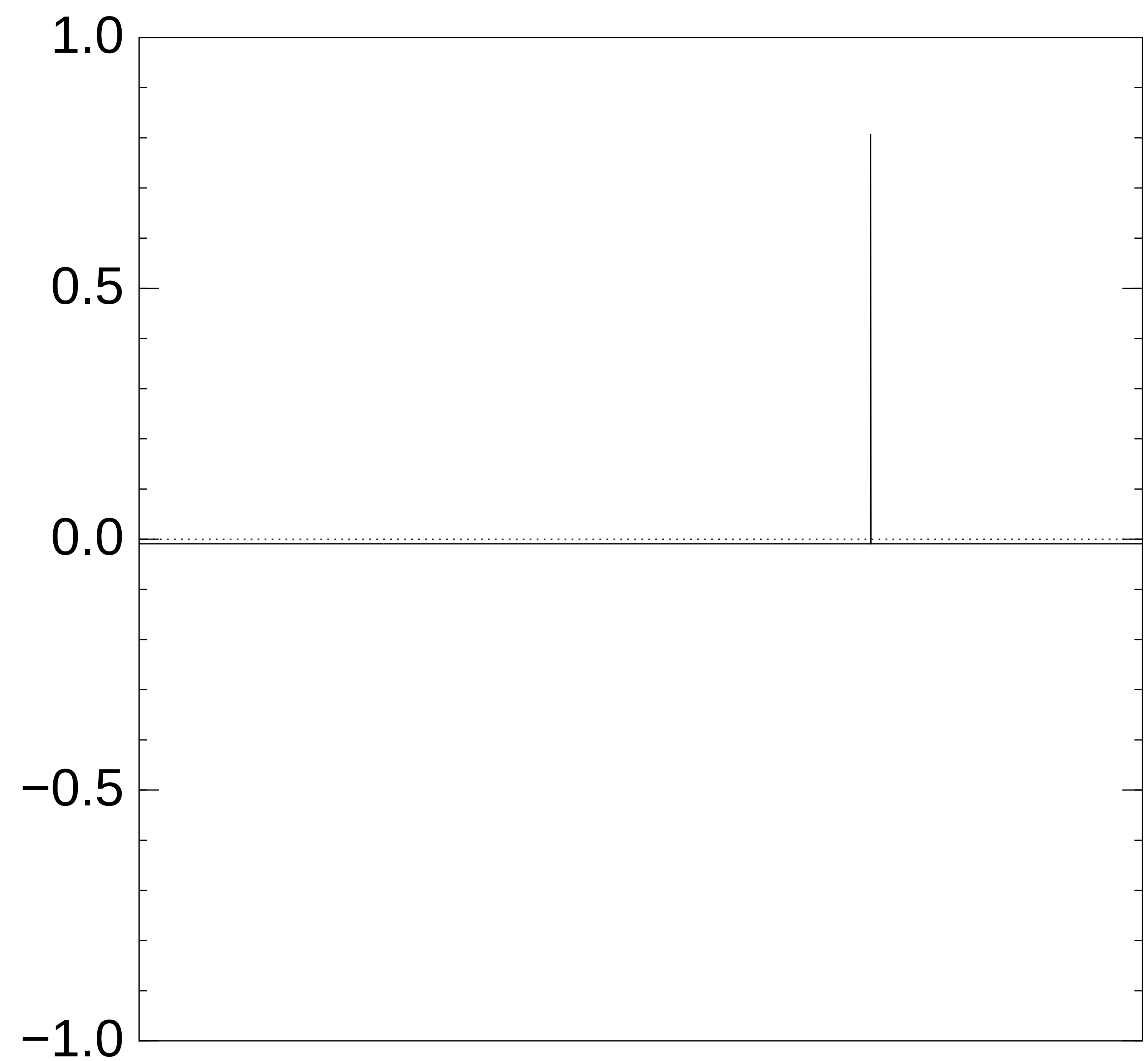
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

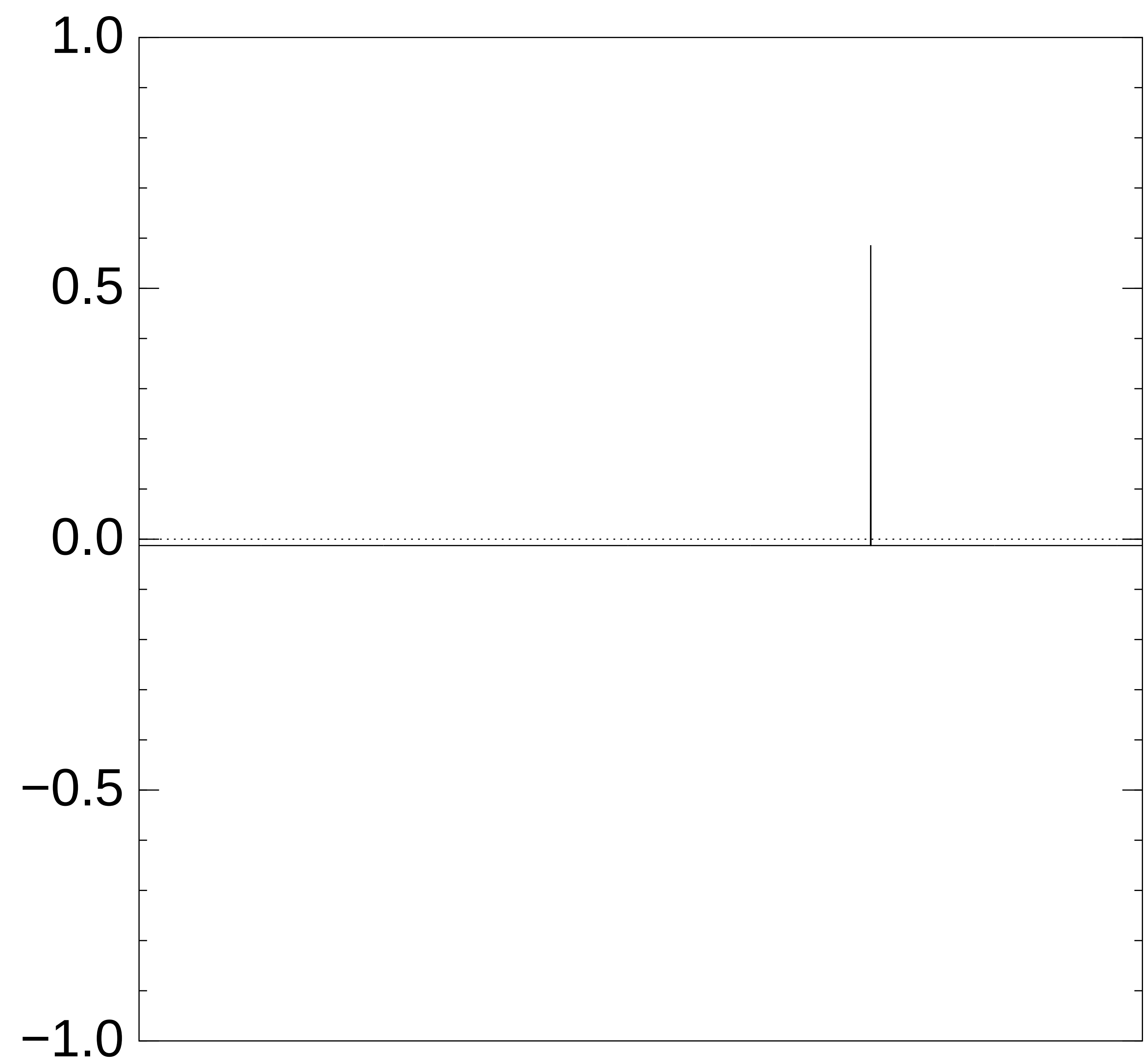
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $80 \times$ (Step 1 + Step 2):



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

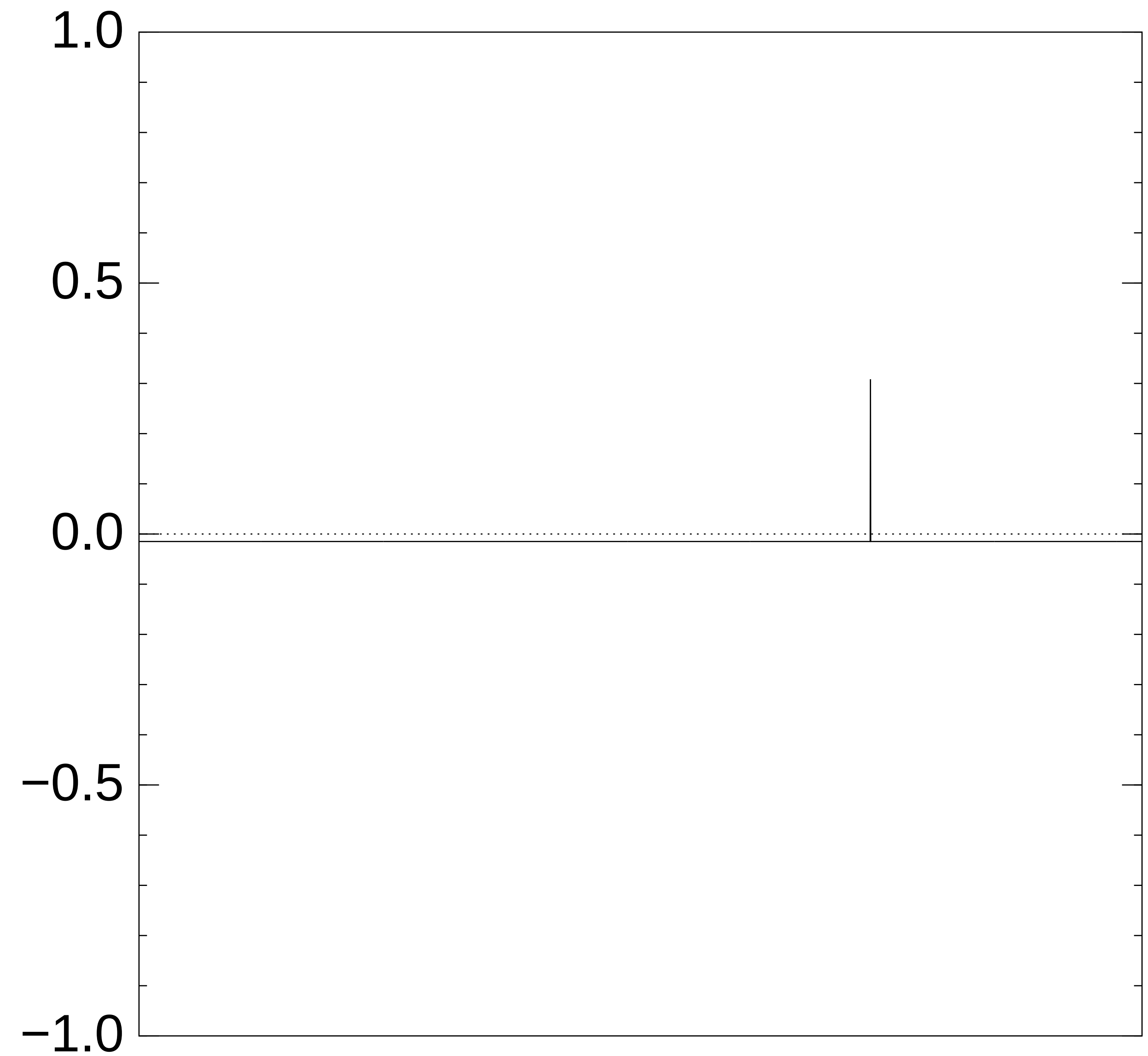
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



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

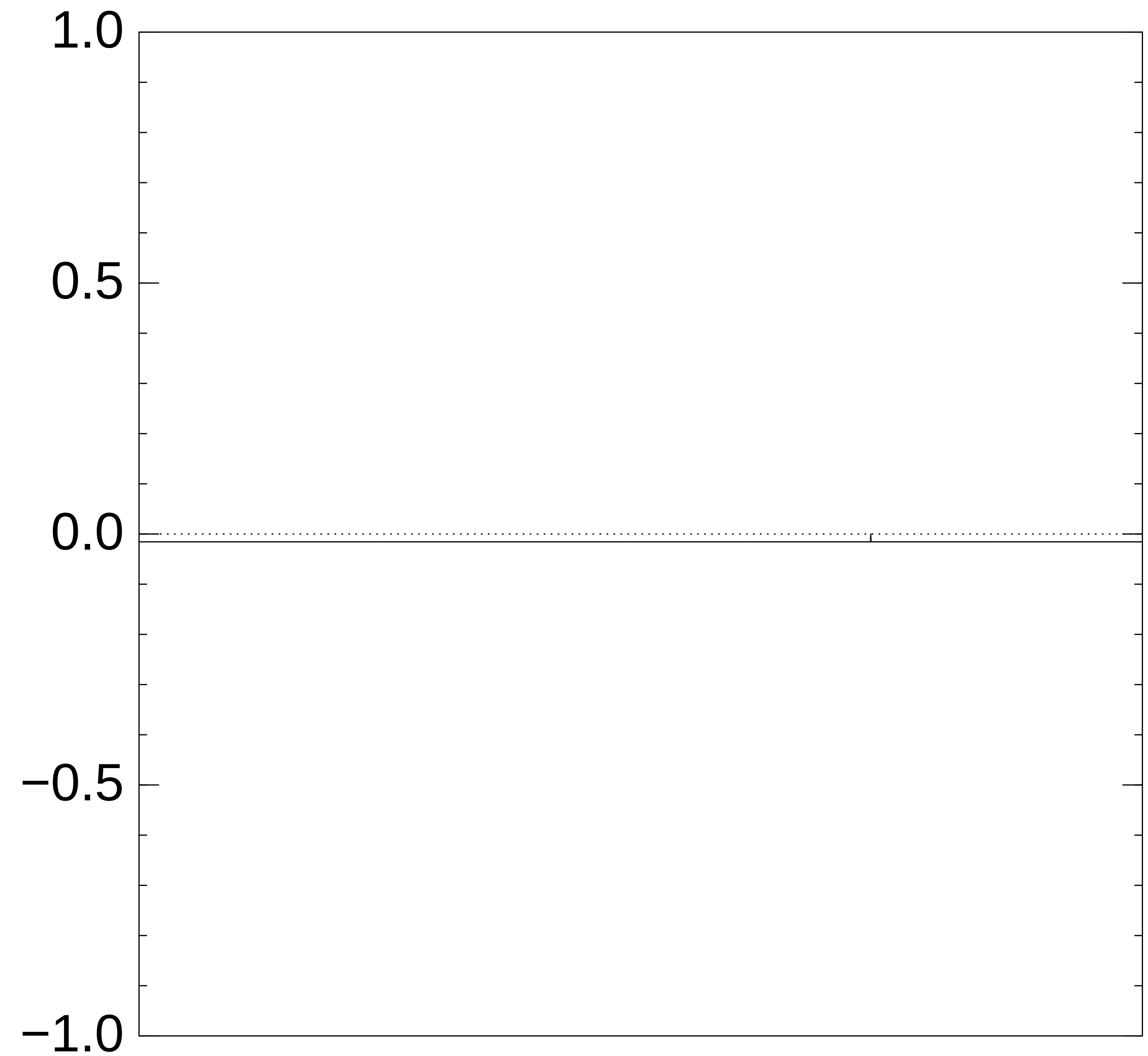
Step 1: Set $a \leftarrow b$ where
$b_q = -a_q$ if $f(q) = 0$,
$b_q = a_q$ otherwise.
This is fast.

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

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

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

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



Very bad stopping point.

om uniform superposition
$\in \{0, 1\}^n$: $a_q = 2^{-n/2}$.

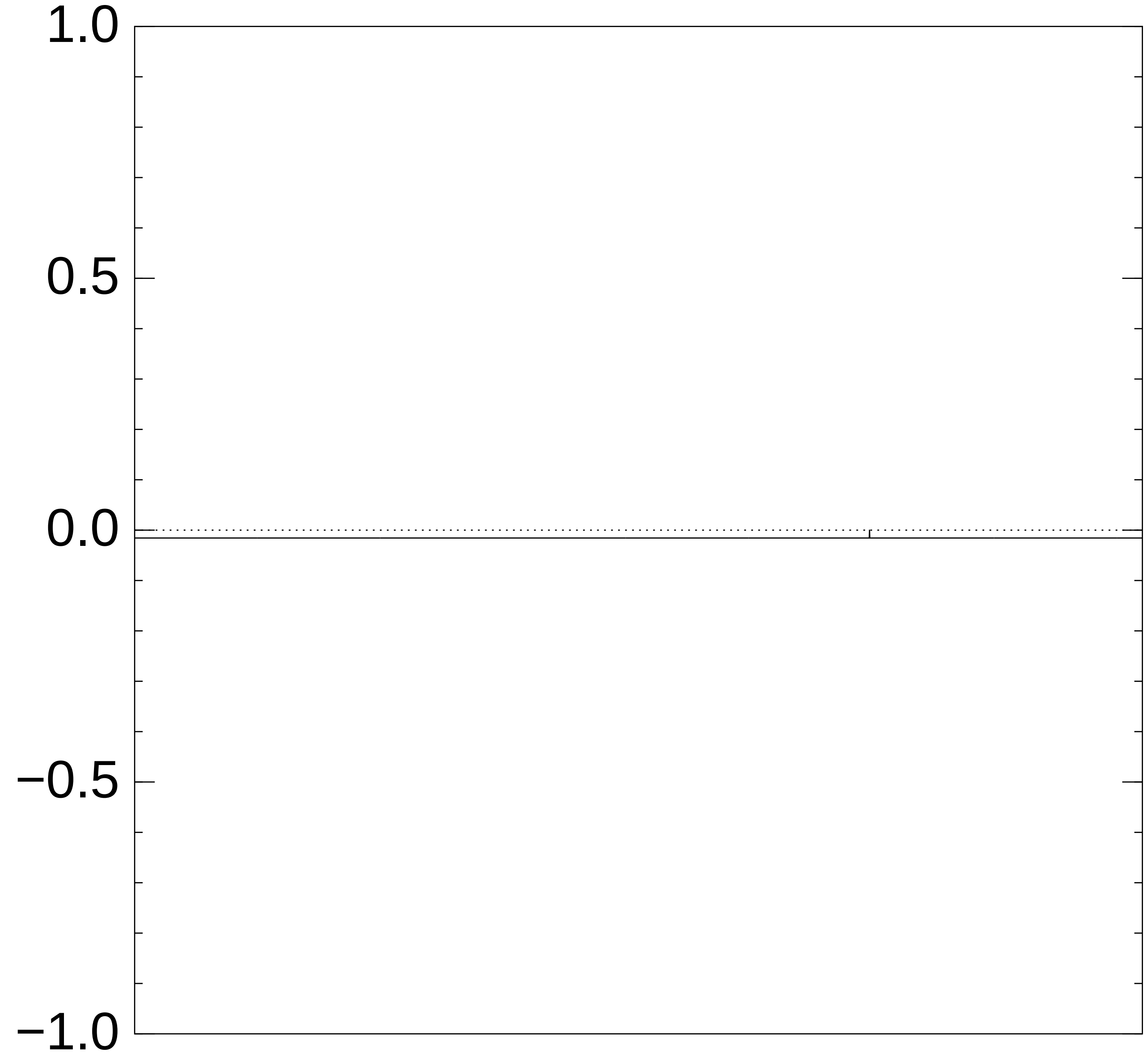
Set $a \leftarrow b$ where
a_q if $f(q) = 0$,
otherwise.
ast.

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

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

the n qubits.
gh probability this finds s.

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



Very bad stopping point.

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

n superposition
: $a_q = 2^{-n/2}$.

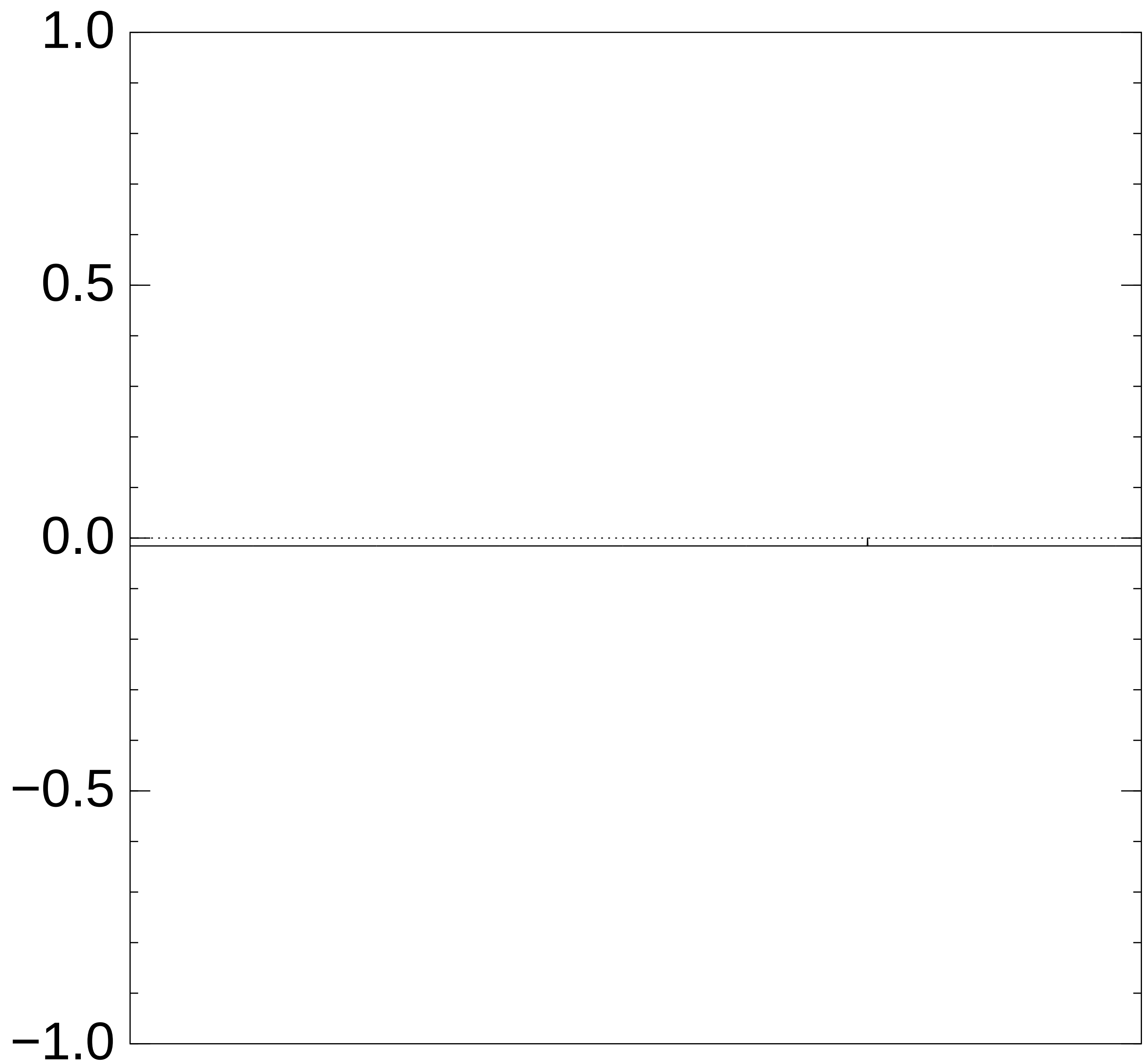
where
$= 0$,
.

iffusion".
ts average.

Step 2
times.

bits.
lity this finds s.

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



Very bad stopping point.

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

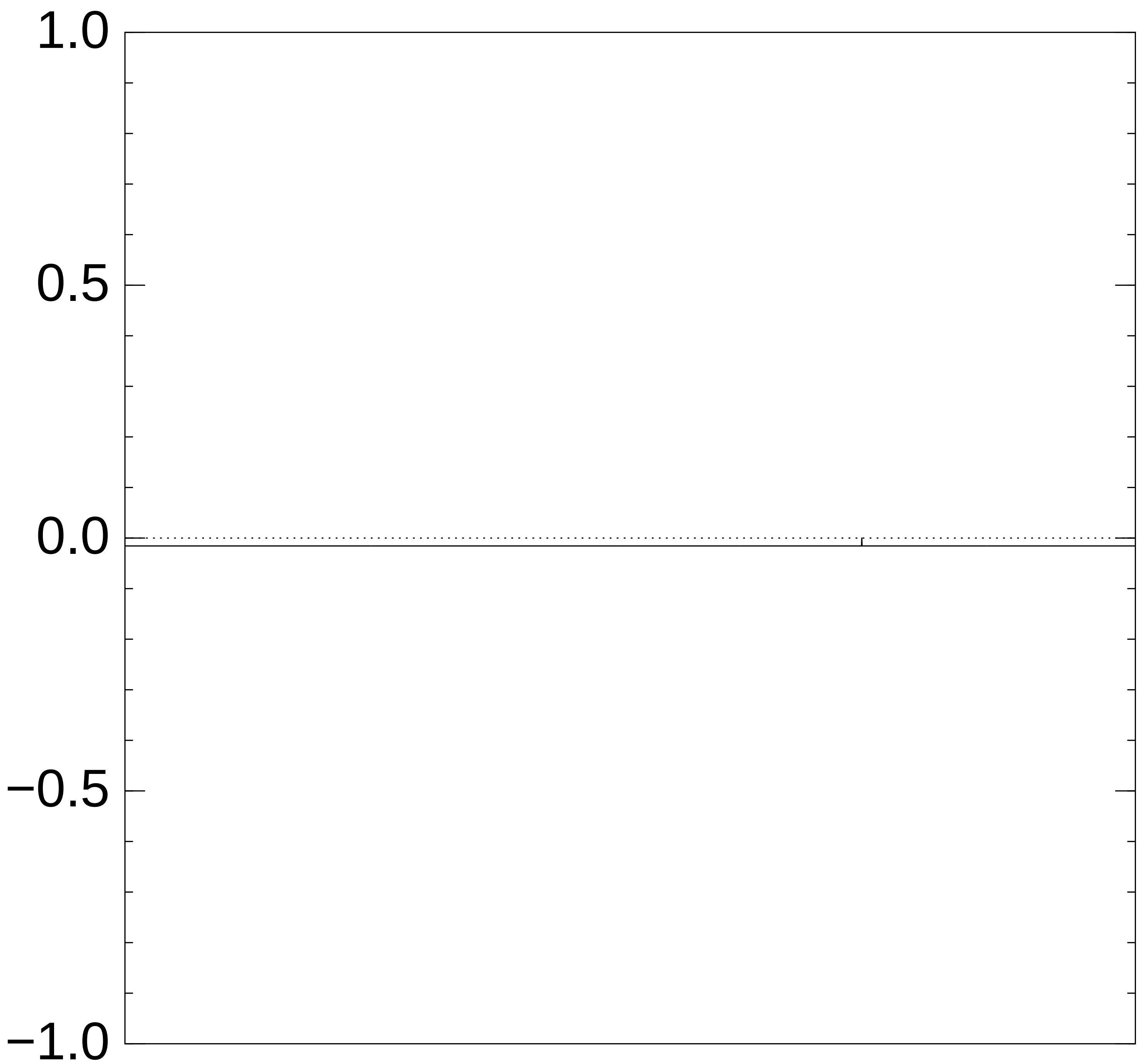
sition

$n/2$.

.

nds s.

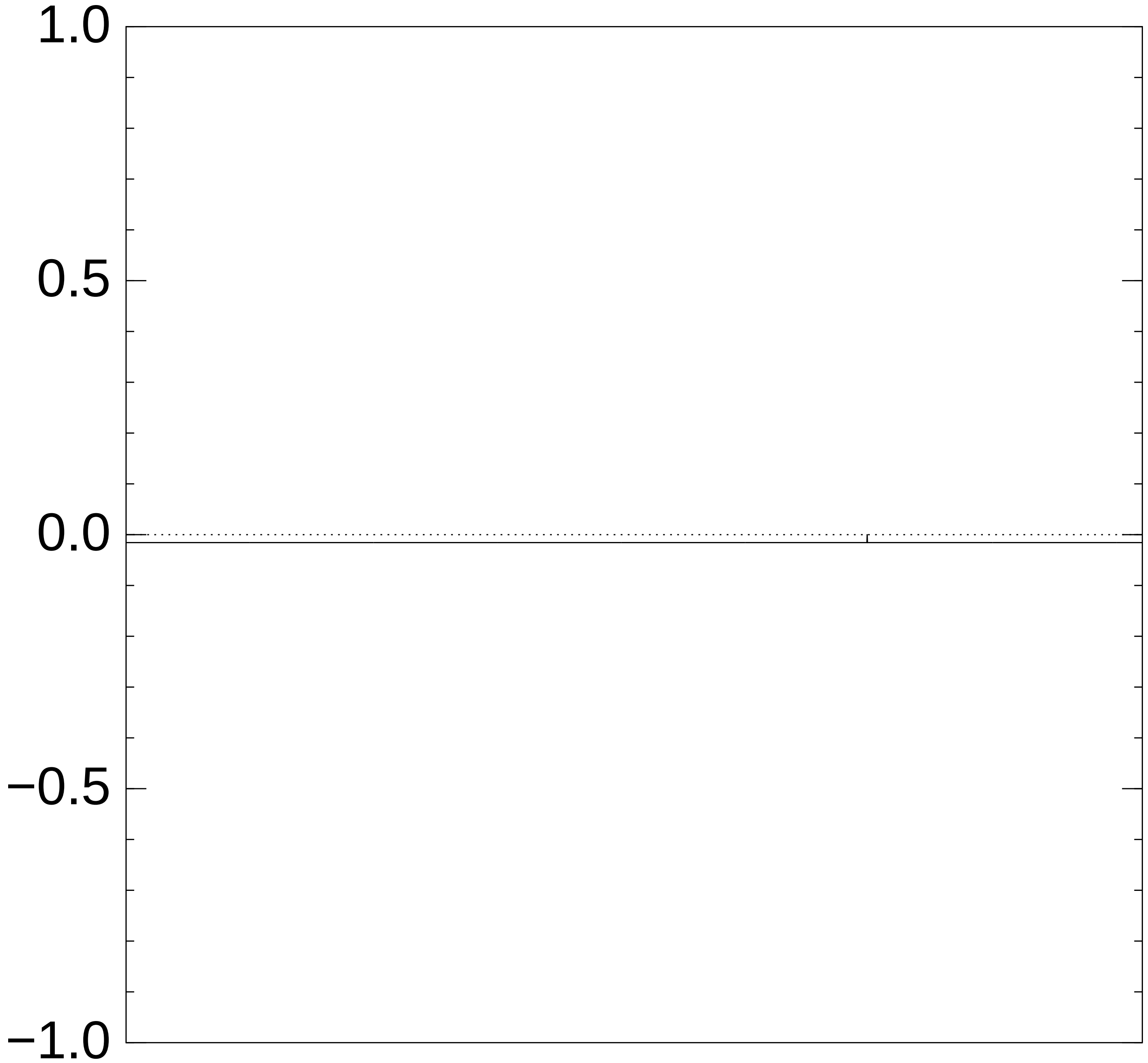
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times (\text{Step } 1 + \text{Step } 2)$:



Very bad stopping point.

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

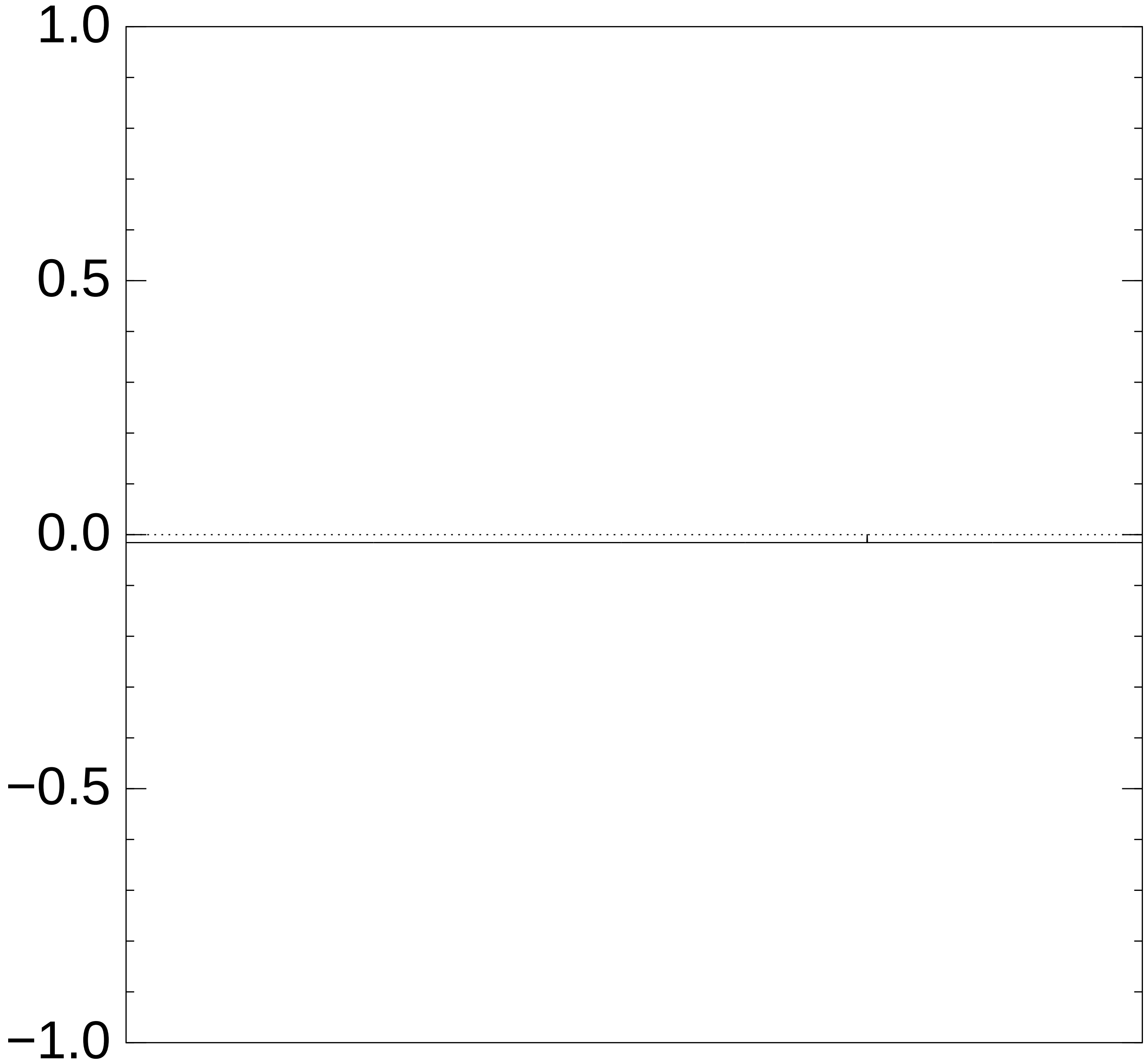
Normalized graph of $q \mapsto a_q$
for an example with $n = 12$
after $100 \times (\text{Step } 1 + \text{Step } 2)$:



Very bad stopping point.

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

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

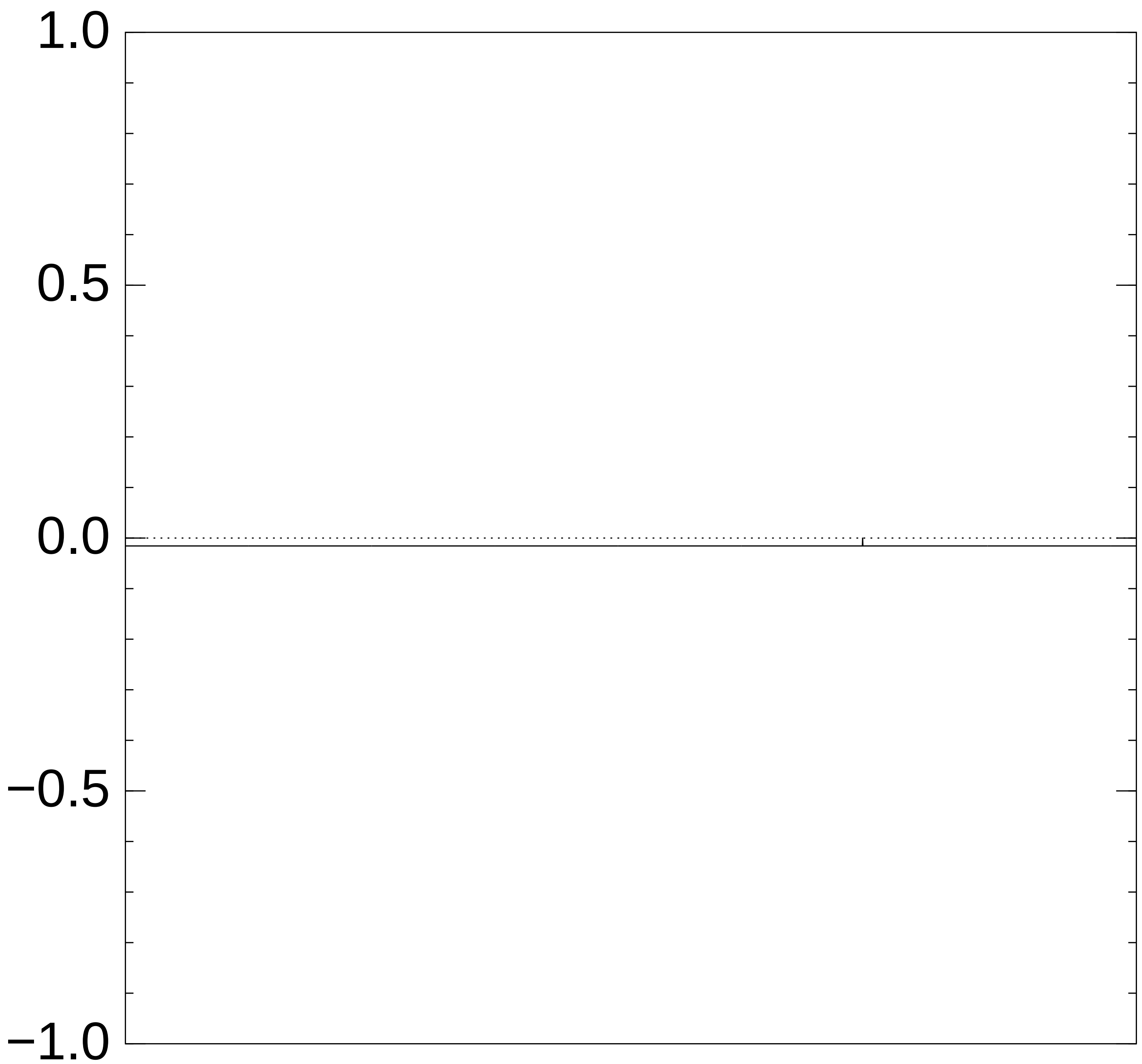


Very bad stopping point.

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

Step $1 +$ Step 2
act linearly on this vector.

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



Very bad stopping point.

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

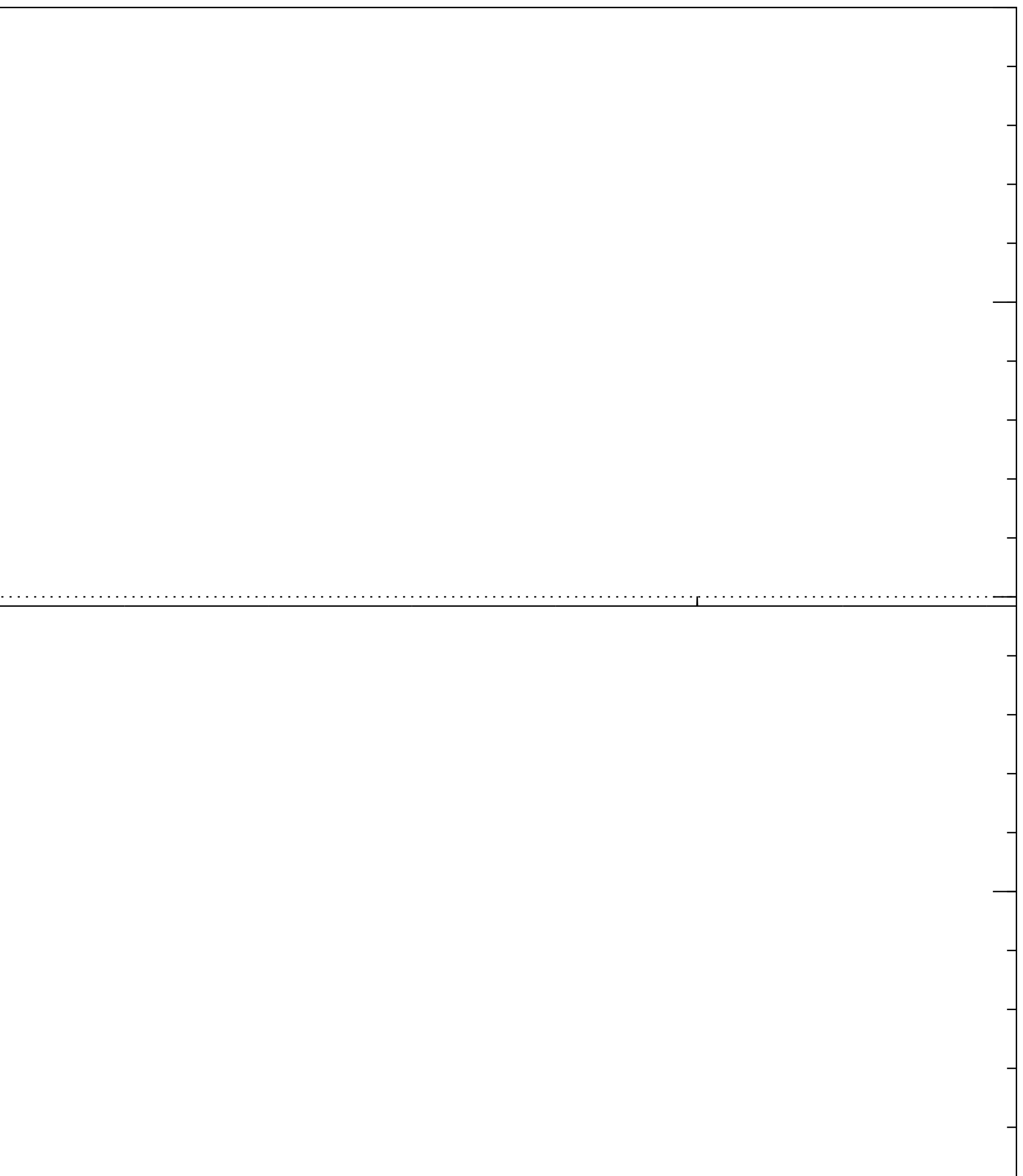
Step 1 + Step 2
act linearly on this vector.

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

zed graph of $q \mapsto a_q$

xample with $n = 12$

0 $\times$ (Step 1 + Step 2):



d stopping point.

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

Step 1 + Step 2
act linearly on this vector.

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

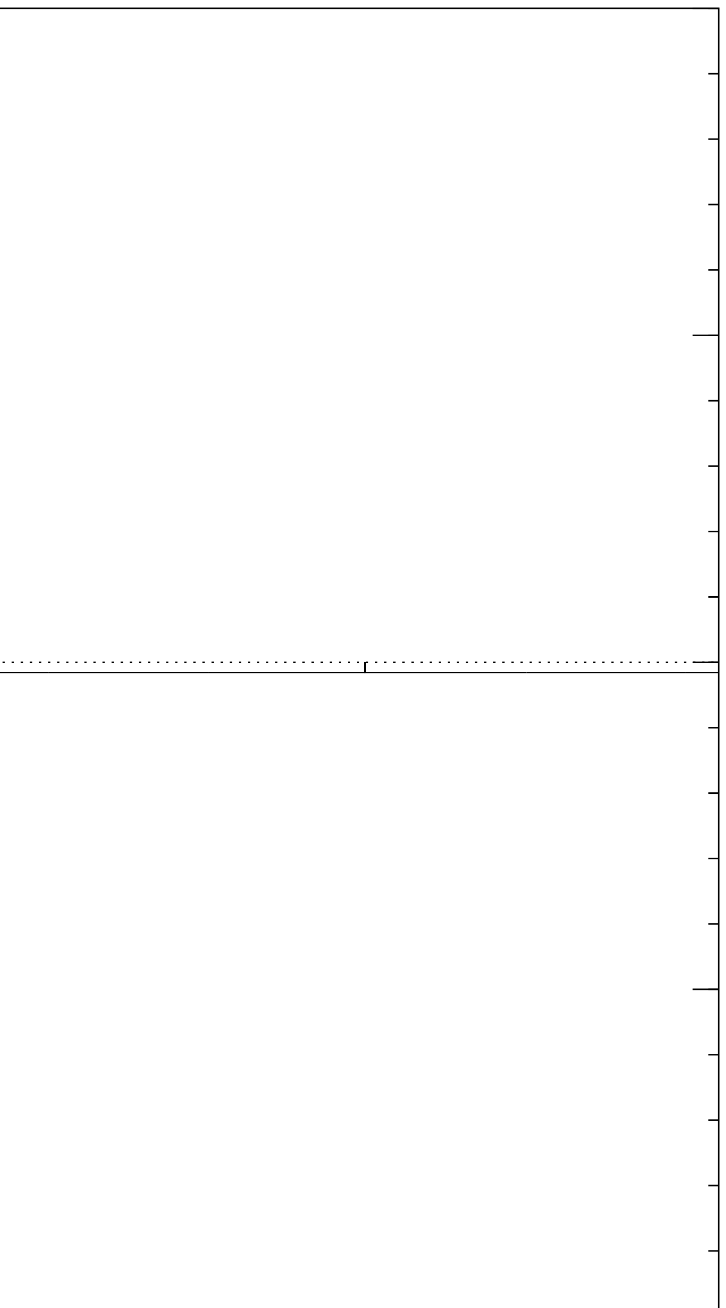
Ambaini

Unique-c
Say f ha
exactly c
i.e., $p \neq$
Problem

of $q \mapsto a_q$

th $n = 12$

$1 +$ Step 2):



point.

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

(1) a_q for roots q;

(2) a_q for non-roots q.

Step 1 + Step 2 act linearly on this vector.

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

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

Ambainis's algorit

Unique-collision-fi

Say f has n-bit in

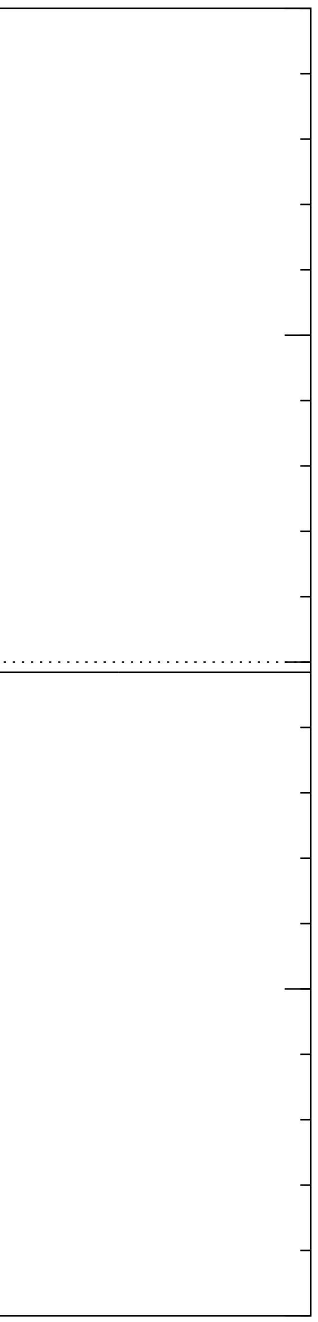
exactly one collisi

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

Problem: find this

q

2):



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

Step 1 + Step 2 act linearly on this vector.

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

Ambainis's algorithm

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

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

Step 1 + Step 2 act linearly on this vector.

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

Ambainis's algorithm

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

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

Step 1 + Step 2 act linearly on this vector.

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

Ambainis's algorithm

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

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

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

Step 1 + Step 2 act linearly on this vector.

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

Ambainis's algorithm

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

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

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

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

+ Step 2
rly on this vector.

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

Ambainis's algorithm

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

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

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

Data str
the gene
the set S
the num

tely described
numbers
licities):

ts q.

s vector.

genvalues
s linear map
lution
s algorithm.
$\approx$1
iterations.

Ambainis's algorithm

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

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

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

Data structure $D($
the generalized co
the set S; the mul
the number of coll

ped

ıp

m.

Ambainis's algorithm

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

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

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

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

Ambainis's algorithm

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

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

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

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

Ambainis's algorithm

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

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

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

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

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

Ambainis's algorithm

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

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

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

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

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

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

s's algorithm

collision-finding problem:
as n-bit inputs,
one collision $\{p, q\}$:
q, $f(p) = f(q)$.
: find this collision.

Define S as
of n-bit strings.
e $f(S)$, sort.

ze to cost r,
probability $\approx(r/2^n)^2$:
a set S of size r.
e $f(S)$, sort.

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

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

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

How the

Start fro
Repeat
Negat
if S
Repea
For
D
For
D

Now hig
that T c
Cost $r +$

hm

nding problem:
puts,
on $\{p, q\}$:
$= f(q)$.
s collision.

as
rings.
rt.

r,
$\approx (r/2^n)^2$:
size r.
rt.

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

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

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

How the quantum

Start from uniform
Repeat $\approx 0.6 \cdot 2^n/$
Negate $a_{S,T}$
if S contains
Repeat $\approx 0.7 \cdot \sqrt{}$
For each T:
Diffuse $a_{S,}$
For each S:
Diffuse $a_{S,}$

Now high probabil
that T contains co
Cost $r + 2^n/\sqrt{r}$.

blem:

2:

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

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

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

How the quantum walk work

Start from uniform superpos
Repeat $\approx 0.6 \cdot 2^n/r$ times:
  Negate $a_{S,T}$
    if S contains collision.
  Repeat $\approx 0.7 \cdot \sqrt{r}$ times:
    For each T:
      Diffuse $a_{S,T}$ across a
    For each S:
      Diffuse $a_{S,T}$ across a

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

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

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

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

How the quantum walk works:

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

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

ucture $D(S)$ capturing
eralized computation:
S; the multiset $f(S)$;
ber of collisions in S.

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

nbainis, simplified 2007
–Nayak–Roland–Santha:
uperposition of states
$D(T))$ with adjacent S, T.
antum walk
ontaining a collision.

How the quantum walk works:

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

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

Classify
$(\#(S \cap$
reduce a
Analyze

e.g. $n =$
0 negati

Pr[class
Pr[class
Pr[class
Pr[class
Pr[class
Pr[class
Pr[class

Right co

S) capturing
mputation:
tiset $f(S)$;
lisions in S.

nove from $D(S)$
adjacent set:
$(S \cap T) = r - 1$.

mplified 2007
oland–Santha:
on of states
adjacent S, T.
lk
a collision.

How the quantum walk works:

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

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

Classify (S, T) acc
$(\#(S \cap \{p, q\}), \#$
reduce a to low-di
Analyze evolution

e.g. $n = 15$, $r = 1$
0 negations and 0

Pr[class $(0, 0)$] $\approx$
Pr[class $(0, 1)$] $\approx$
Pr[class $(1, 0)$] $\approx$
Pr[class $(1, 1)$] $\approx$
Pr[class $(1, 2)$] $\approx$
Pr[class $(2, 1)$] $\approx$
Pr[class $(2, 2)$] $\approx$

Right column is si

ing

:

;

S.

$D(S)$

set:

$r-1$.

007

ntha:

es

S, T.

.

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vec

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

0 negations and 0 diffusions

$\Pr[\text{class } (0,0)] \approx 0.938; +$

$\Pr[\text{class } (0,1)] \approx 0.000; +$

$\Pr[\text{class } (1,0)] \approx 0.000; +$

$\Pr[\text{class } (1,1)] \approx 0.060; +$

$\Pr[\text{class } (1,2)] \approx 0.000; +$

$\Pr[\text{class } (2,1)] \approx 0.000; +$

$\Pr[\text{class } (2,2)] \approx 0.001; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

0 negations and 0 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.938; +$

$\Pr[\text{class } (0, 1)] \approx 0.000; +$

$\Pr[\text{class } (1, 0)] \approx 0.000; +$

$\Pr[\text{class } (1, 1)] \approx 0.060; +$

$\Pr[\text{class } (1, 2)] \approx 0.000; +$

$\Pr[\text{class } (2, 1)] \approx 0.000; +$

$\Pr[\text{class } (2, 2)] \approx 0.001; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

1 negation and 46 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.935; +$

$\Pr[\text{class } (0, 1)] \approx 0.000; +$

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

$\Pr[\text{class } (1, 1)] \approx 0.057; +$

$\Pr[\text{class } (1, 2)] \approx 0.000; +$

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

$\Pr[\text{class } (2, 2)] \approx 0.008; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

2 negations and 92 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.918; +$

$\Pr[\text{class } (0,1)] \approx 0.001; +$

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

$\Pr[\text{class } (1,1)] \approx 0.059; +$

$\Pr[\text{class } (1,2)] \approx 0.001; +$

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

$\Pr[\text{class } (2,2)] \approx 0.022; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

3 negations and 138 diffusions:

Pr[class $(0, 0)$] $\approx 0.897; +$

Pr[class $(0, 1)$] $\approx 0.001; +$

Pr[class $(1, 0)$] $\approx 0.000; -$

Pr[class $(1, 1)$] $\approx 0.058; +$

Pr[class $(1, 2)$] $\approx 0.002; +$

Pr[class $(2, 1)$] $\approx 0.000; +$

Pr[class $(2, 2)$] $\approx 0.042; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

4 negations and 184 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.873; +$

$\Pr[\text{class } (0, 1)] \approx 0.001; +$

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

$\Pr[\text{class } (1, 1)] \approx 0.054; +$

$\Pr[\text{class } (1, 2)] \approx 0.002; +$

$\Pr[\text{class } (2, 1)] \approx 0.000; +$

$\Pr[\text{class } (2, 2)] \approx 0.070; +$

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

How the quantum walk works:

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

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

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

e.g. $n = 15$, $r = 1024$, after
5 negations and 230 diffusions:

$\Pr[\text{class }(0,0)] \approx 0.838; +$
$\Pr[\text{class }(0,1)] \approx 0.001; +$
$\Pr[\text{class }(1,0)] \approx 0.001; -$
$\Pr[\text{class }(1,1)] \approx 0.054; +$
$\Pr[\text{class }(1,2)] \approx 0.003; +$
$\Pr[\text{class }(2,1)] \approx 0.000; +$
$\Pr[\text{class }(2,2)] \approx 0.104; +$

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

How the quantum walk works:

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

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

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

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

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

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

7 negations and 322 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.758; +$

$\Pr[\text{class } (0, 1)] \approx 0.002; +$

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

$\Pr[\text{class } (1, 1)] \approx 0.047; +$

$\Pr[\text{class } (1, 2)] \approx 0.007; +$

$\Pr[\text{class } (2, 1)] \approx 0.000; +$

$\Pr[\text{class } (2, 2)] \approx 0.184; +$

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

How the quantum walk works:

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

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

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

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

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

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

How the quantum walk works:

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

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

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

e.g. $n = 15$, $r = 1024$, after
9 negations and 414 diffusions:

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

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

10 negations and 460 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.606; +$

$\Pr[\text{class } (0,1)] \approx 0.003; +$

$\Pr[\text{class } (1,0)] \approx 0.002; -$

$\Pr[\text{class } (1,1)] \approx 0.037; +$

$\Pr[\text{class } (1,2)] \approx 0.013; +$

$\Pr[\text{class } (2,1)] \approx 0.000; +$

$\Pr[\text{class } (2,2)] \approx 0.338; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

11 negations and 506 diffusions:

Pr[class $(0, 0)$] $\approx 0.547; +$

Pr[class $(0, 1)$] $\approx 0.004; +$

Pr[class $(1, 0)$] $\approx 0.003; -$

Pr[class $(1, 1)$] $\approx 0.036; +$

Pr[class $(1, 2)$] $\approx 0.015; +$

Pr[class $(2, 1)$] $\approx 0.001; +$

Pr[class $(2, 2)$] $\approx 0.394; +$

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

How the quantum walk works:

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

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

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

e.g. $n = 15$, $r = 1024$, after
12 negations and 552 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.491; +$
$\Pr[\text{class } (0, 1)] \approx 0.004; +$
$\Pr[\text{class } (1, 0)] \approx 0.003; -$
$\Pr[\text{class } (1, 1)] \approx 0.032; +$
$\Pr[\text{class } (1, 2)] \approx 0.014; +$
$\Pr[\text{class } (2, 1)] \approx 0.001; +$
$\Pr[\text{class } (2, 2)] \approx 0.455; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

13 negations and 598 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.436; +$

$\Pr[\text{class } (0,1)] \approx 0.005; +$

$\Pr[\text{class } (1,0)] \approx 0.003; -$

$\Pr[\text{class } (1,1)] \approx 0.026; +$

$\Pr[\text{class } (1,2)] \approx 0.017; +$

$\Pr[\text{class } (2,1)] \approx 0.000; +$

$\Pr[\text{class } (2,2)] \approx 0.513; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S,T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

14 negations and 644 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.377; +$

$\Pr[\text{class } (0,1)] \approx 0.006; +$

$\Pr[\text{class } (1,0)] \approx 0.004; -$

$\Pr[\text{class } (1,1)] \approx 0.025; +$

$\Pr[\text{class } (1,2)] \approx 0.022; +$

$\Pr[\text{class } (2,1)] \approx 0.001; +$

$\Pr[\text{class } (2,2)] \approx 0.566; +$

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

How the quantum walk works:

Start from uniform superposition.

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

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

Now high probability that T contains collision.

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

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

Analyze evolution of this vector.

e.g. $n = 15$, $r = 1024$, after 15 negations and 690 diffusions:

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

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

How the quantum walk works:

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

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

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

e.g. $n = 15$, $r = 1024$, after
16 negations and 736 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.270; +$
$\Pr[\text{class } (0,1)] \approx 0.006; +$
$\Pr[\text{class } (1,0)] \approx 0.005; -$
$\Pr[\text{class } (1,1)] \approx 0.017; +$
$\Pr[\text{class } (1,2)] \approx 0.022; +$
$\Pr[\text{class } (2,1)] \approx 0.001; +$
$\Pr[\text{class } (2,2)] \approx 0.680; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

17 negations and 782 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.218; +$

$\Pr[\text{class } (0, 1)] \approx 0.007; +$

$\Pr[\text{class } (1, 0)] \approx 0.005; -$

$\Pr[\text{class } (1, 1)] \approx 0.015; +$

$\Pr[\text{class } (1, 2)] \approx 0.024; +$

$\Pr[\text{class } (2, 1)] \approx 0.001; +$

$\Pr[\text{class } (2, 2)] \approx 0.730; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

18 negations and 828 diffusions:

Pr[class $(0, 0)$] $\approx 0.172; +$

Pr[class $(0, 1)$] $\approx 0.006; +$

Pr[class $(1, 0)$] $\approx 0.005; -$

Pr[class $(1, 1)$] $\approx 0.011; +$

Pr[class $(1, 2)$] $\approx 0.029; +$

Pr[class $(2, 1)$] $\approx 0.001; +$

Pr[class $(2, 2)$] $\approx 0.775; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

19 negations and 874 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.131; +$

$\Pr[\text{class } (0,1)] \approx 0.007; +$

$\Pr[\text{class } (1,0)] \approx 0.006; -$

$\Pr[\text{class } (1,1)] \approx 0.008; +$

$\Pr[\text{class } (1,2)] \approx 0.030; +$

$\Pr[\text{class } (2,1)] \approx 0.002; +$

$\Pr[\text{class } (2,2)] \approx 0.816; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

20 negations and 920 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.093; +$

$\Pr[\text{class } (0,1)] \approx 0.007; +$

$\Pr[\text{class } (1,0)] \approx 0.007; -$

$\Pr[\text{class } (1,1)] \approx 0.007; +$

$\Pr[\text{class } (1,2)] \approx 0.027; +$

$\Pr[\text{class } (2,1)] \approx 0.002; +$

$\Pr[\text{class } (2,2)] \approx 0.857; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

21 negations and 966 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.062; +$

$\Pr[\text{class } (0,1)] \approx 0.007; +$

$\Pr[\text{class } (1,0)] \approx 0.006; -$

$\Pr[\text{class } (1,1)] \approx 0.004; +$

$\Pr[\text{class } (1,2)] \approx 0.030; +$

$\Pr[\text{class } (2,1)] \approx 0.001; +$

$\Pr[\text{class } (2,2)] \approx 0.890; +$

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

How the quantum walk works:

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

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

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

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

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

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

23 negations and 1058 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.017; +$

$\Pr[\text{class } (0, 1)] \approx 0.008; +$

$\Pr[\text{class } (1, 0)] \approx 0.007; -$

$\Pr[\text{class } (1, 1)] \approx 0.002; +$

$\Pr[\text{class } (1, 2)] \approx 0.034; +$

$\Pr[\text{class } (2, 1)] \approx 0.002; +$

$\Pr[\text{class } (2, 2)] \approx 0.930; +$

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

How the quantum walk works:

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

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

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

e.g. $n = 15$, $r = 1024$, after
24 negations and 1104 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.005; +$
$\Pr[\text{class } (0,1)] \approx 0.007; +$
$\Pr[\text{class } (1,0)] \approx 0.007; -$
$\Pr[\text{class } (1,1)] \approx 0.000; +$
$\Pr[\text{class } (1,2)] \approx 0.030; +$
$\Pr[\text{class } (2,1)] \approx 0.002; +$
$\Pr[\text{class } (2,2)] \approx 0.948; +$

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

How the quantum walk works:

Start from uniform superposition.

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

Negate $a_{S,T}$

if S contains collision.

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

For each T:

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

For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

25 negations and 1150 diffusions:

$\Pr[\text{class } (0,0)] \approx 0.000; +$

$\Pr[\text{class } (0,1)] \approx 0.008; +$

$\Pr[\text{class } (1,0)] \approx 0.008; -$

$\Pr[\text{class } (1,1)] \approx 0.000; +$

$\Pr[\text{class } (1,2)] \approx 0.031; +$

$\Pr[\text{class } (2,1)] \approx 0.001; +$

$\Pr[\text{class } (2,2)] \approx 0.952; +$

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

How the quantum walk works:

Start from uniform superposition.

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

  Negate $a_{S,T}$

    if S contains collision.

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

    For each T:

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

    For each S:

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

Now high probability

that T contains collision.

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

Classify (S, T) according to

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

reduce a to low-dim vector.

Analyze evolution of this vector.

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

26 negations and 1196 diffusions:

$\Pr[\text{class } (0, 0)] \approx 0.002; -$

$\Pr[\text{class } (0, 1)] \approx 0.008; +$

$\Pr[\text{class } (1, 0)] \approx 0.008; -$

$\Pr[\text{class } (1, 1)] \approx 0.000; -$

$\Pr[\text{class } (1, 2)] \approx 0.035; +$

$\Pr[\text{class } (2, 1)] \approx 0.002; +$

$\Pr[\text{class } (2, 2)] \approx 0.945; +$

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

How the quantum walk works:

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

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

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

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

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

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

e quantum walk works:

om uniform superposition.

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

ze $a_{S,T}$

contains collision.

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

each T:

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

each S:

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

h probability

contains collision.

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

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

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

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

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

Data str

Moving

dominat

of f if f

But usua

walk works:

n superposition.

r times:

collision.

$\sqrt{r}$ times:

$_T$ across all S.

$_T$ across all T.

ity

ollision.

Optimize: $2^{2n/3}$.

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

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

$\Pr[\text{class } (0,0)] \approx 0.011; -$

$\Pr[\text{class } (0,1)] \approx 0.007; +$

$\Pr[\text{class } (1,0)] \approx 0.007; -$

$\Pr[\text{class } (1,1)] \approx 0.001; -$

$\Pr[\text{class } (1,2)] \approx 0.034; +$

$\Pr[\text{class } (2,1)] \approx 0.003; +$

$\Pr[\text{class } (2,2)] \approx 0.938; +$

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

Data structures

Moving from $D(S$

dominated by $O(1$

of f if f is extrem

But usually f is no

ks:

sition.

ll S.

ll T.

$2^{2n/3}$.

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

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

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

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

Data structures

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

But usually f is not so slow.

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

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

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

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

Data structures

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

But usually f is not so slow.

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

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

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

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

Data structures

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

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

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

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

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

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

Data structures

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

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

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

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

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

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

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

Data structures

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

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

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

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

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

15, $r = 1024$, after
tions and 1242 diffusions:

$(0, 0)] \approx 0.011; -$
$(0, 1)] \approx 0.007; +$
$(1, 0)] \approx 0.007; -$
$(1, 1)] \approx 0.001; -$
$(1, 2)] \approx 0.034; +$
$(2, 1)] \approx 0.003; +$
$(2, 2)] \approx 0.938; +$

lumn is sign of $a_{S,T}$.

Data structures

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

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

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

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

Major pr
depends
S. Algo

Need his

cording to
$(T \cap \{p, q\}))$;
m vector.
of this vector.

024, after
1242 diffusions:

$0.011; -$
$0.007; +$
$0.007; -$
$0.001; -$
$0.034; +$
$0.003; +$
$0.938; +$

gn of $a_{S,T}$.

Data structures

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

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

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

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

Major problem: ha
depends on history
S. Algorithm fails

Need history-indep

}));

ctor.

sions:

.

Data structures

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

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

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

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

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

Need history-independent D

Data structures

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

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

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

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

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

Need history-independent $D(S)$.

Data structures

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

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

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

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

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

Need history-independent $D(S)$.

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

Data structures

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

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

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

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

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

uctures

from $D(S)$ to $D(T)$:
ed by $O(1)$ evaluations
is extremely slow.

ally f is not so slow.
t S and multiset $f(S)$
hash tables?

roblem: time to hash S
for some sets S.

domize hash function
arter–Wegman),
cify big enough time for
gorithm to be reliable.

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

The $2^{2n/}$
cheap ra
Justified

) to $D(T)$:
) evaluations
ely slow.

ot so slow.
ultiset $f(S)$
es?

me to hash S
ets S.

sh function
gman),
ough time for
o be reliable.

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

The $2^{2n/3}$ analysis
cheap random acc
Justified by simpli

:
ons
.
$S)$
h S
n
for
le.

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

Can we move data using energy sublinear in distance moved?

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

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

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

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

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

Need history-independent $D(S)$.

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

2013 Bernstein–Jeffery–Lange–Meurer: radix tree.

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

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

roblem: hash table
on history, not just on
rithm fails horribly.

story-independent $D(S)$.

nbainis: "combination
h table and a skip list".
pages of analysis.

rnstein–Jeffery–Lange–
radix tree.

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

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many cl
don't se
e.g. 200
fastest a
random-
1994 va

ash table
y, not just on
horribly.

pendent $D(S)$.

combination
d a skip list".
nalysis.

effery–Lange–
e.

e: Left
$(0, x) \in S\}$
t subtree stores
nonempty.

Caveats

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

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

I recommend algorithm analysis
on 2-dim mesh of tiny processors:
e.g. 0.472 for MQ (vs. 0.462)
from 2017 Bernstein–Yang.

Many claimed qua
don't seem to exis
e.g. 2009 Bernstei
fastest algorithm k
random-collision se
1994 van Oorscho

on

(S).

on
st".

ge–

$S\}$
stores
.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum spe
don't seem to exist in this m
e.g. 2009 Bernstein analysis
fastest algorithm known for
random-collision search is
1994 van Oorschot–Wiener.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Caveats

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

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

I recommend algorithm analysis on 2-dim mesh of tiny processors: e.g. 0.472 for MQ (vs. 0.462) from 2017 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

/3 analysis assumes
ndom access to memory.
by simplicity, not realism.

move data using energy
r in distance moved?
el presentation says that
8 bytes on wire at 22nm
.20 pJ per 5mm.
pread. Fibers lose. etc.

mend algorithm analysis
n mesh of tiny processors:
72 for MQ (vs. 0.462)
17 Bernstein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

Backgro

s assumes
ess to memory.
city, not realism.

a using energy
ce moved?
ation says that
n wire at 22nm
r 5mm.
pers lose. etc.

rithm analysis
tiny processors:
(vs. 0.462)
ein–Yang.

Many claimed quantum speedups don't seem to exist in this model. e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

Background slides

mory.
ealism.

ergy
?
that
2nm

etc.

ysis
essors:
2)

Many claimed quantum speedups don't seem to exist in this model.
e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

Background slides . . .

Many claimed quantum speedups don't seem to exist in this model.
e.g. 2009 Bernstein analysis: fastest algorithm known for random-collision search is 1994 van Oorschot–Wiener.

Further obstacles to Grover:

- Parallelization reduces speedup. $D\times$ speedup needs depth D.
- Reversibility is expensive.
- Quantum ops are expensive.

Grover risk to cryptography is much smaller than Shor risk.

Background slides . . .

aimed quantum speedups
em to exist in this model.
9 Bernstein analysis:
lgorithm known for
collision search is
n Oorschot–Wiener.

obstacles to Grover:

elization reduces speedup.
peedup needs depth D.

sibility is expensive.

um ops are expensive.

isk to cryptography
smaller than Shor risk.

Background slides . . .

What do

"Quantu
means a
a quantu

i.e. a seq
where ea
in a qua
supporte

How do
instruct
comput

ntum speedups
t in this model.
n analysis:
known for
earch is
t–Wiener.

to Grover:

educes speedup.
eds depth D.

xpensive.

e expensive.

otography
han Shor risk.

Background slides . . .

What do quantum

"Quantum algorit
means an algorith
a quantum compu

i.e. a sequence of
where each instruc
in a quantum com
supported instruct

**How do we know
instructions a qu
computer will su**

edups

nodel.

t

eedup.

D.

ve.

isk.

Background slides . . .

What do quantum computer

"Quantum algorithm"
means an algorithm that
a quantum computer can ru

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

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

Background slides . . .

What do quantum computers do?

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

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

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

und slides . . .

What do quantum computers do?

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

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

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

Quantur
contains
can effic
"NOT g
"control

. . .

What do quantum computers do?

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

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

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

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

What do quantum computers do?

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

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

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

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

What do quantum computers do?

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

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

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

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

What do quantum computers do?

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

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

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

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

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

What do quantum computers do?

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

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

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

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

What do quantum computers do?

"Quantum algorithm"
means an algorithm that
a quantum computer can run.

i.e. a sequence of instructions,
where each instruction is
in a quantum computer's
supported instruction set.

How do we know which instructions a quantum computer will support?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
. . . "Simon's algorithm";
. . . "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

o quantum computers do?

um algorithm"
n algorithm that
um computer can run.

quence of instructions,
ach instruction is
ntum computer's
ed instruction set.

**we know which
ions a quantum
er will support?**

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
. . . "Simon's algorithm";
. . . "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantur
stores a
efficientl
laws of c
with as r

This is t
quantum
by 1982
physics v

computers do?

hm"
m that
ter can run.

instructions,
ction is
puter's
ion set.

which
antum
pport?

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "*T* gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantum compute
stores a simulated
efficiently simulate
laws of quantum p
with as much accu

This is the origina
quantum compute
by 1982 Feynman
physics with comp

rs do?

n.

ns,

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantum computer type 2 (
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as de

This is the original concept
quantum computers introdu
by 1982 Feynman "Simulati
physics with computers".

Quantum computer type 1 (QC1):
contains many “qubits”;
can efficiently perform
“NOT gate”, “Hadamard gate”,
“controlled NOT gate”, “T gate”.

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute “Toffoli gate”;
... “Simon’s algorithm”;
... “Shor’s algorithm”; etc.

General belief: Traditional CPU isn’t QC1; e.g. can’t factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

Quantum computer type 1 (QC1):
contains many "qubits";
can efficiently perform
"NOT gate", "Hadamard gate",
"controlled NOT gate", "T gate".

Making these instructions work is the main goal of quantum-computer engineering.

Combine these instructions
to compute "Toffoli gate";
... "Simon's algorithm";
... "Shor's algorithm"; etc.

General belief: Traditional CPU
isn't QC1; e.g. can't factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman "Simulating
physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
"Quantum algorithms for
quantum field theories".

m computer type 1 (QC1):
many "qubits";
iently perform
ate", "Hadamard gate",
led NOT gate", "*T* gate".

these instructions work
nain goal of quantum-
er engineering.

e these instructions
ute "Toffoli gate";
non's algorithm";
or's algorithm"; etc.

belief: Traditional CPU
1; e.g. can't factor quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantur
efficientl
that any
compute

er type 1 (QC1):
ubits";
form
damard gate",
gate", "T gate".

structions work
of quantum-
ering.

tructions
oli gate";
rithm";
thm"; etc.

aditional CPU
n't factor quickly.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum compute
efficiently compute
that any possible
computer can com

(QC1):

ate”,
gate”.

work
um-

CPU
quickly.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman “Simulating physics with computers”.

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
“Quantum algorithms for
quantum field theories”.

Quantum computer type 3 (
efficiently computes anythin
that any possible physical
computer can compute effici

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman "Simulating
physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
"Quantum algorithms for
quantum field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

Quantum computer type 2 (QC2): stores a simulated universe; efficiently simulates the laws of quantum physics with as much accuracy as desired.

This is the original concept of quantum computers introduced by 1982 Feynman "Simulating physics with computers".

General belief: any QC1 is a QC2. Partial proof: see, e.g., 2011 Jordan–Lee–Preskill "Quantum algorithms for quantum field theories".

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

Quantum computer type 2 (QC2):
stores a simulated universe;
efficiently simulates the
laws of quantum physics
with as much accuracy as desired.

This is the original concept of
quantum computers introduced
by 1982 Feynman "Simulating
physics with computers".

General belief: any QC1 is a QC2.
Partial proof: see, e.g.,
2011 Jordan–Lee–Preskill
"Quantum algorithms for
quantum field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

m computer type 2 (QC2):
simulated universe;
y simulates the
quantum physics
much accuracy as desired.

he original concept of
n computers introduced
Feynman "Simulating
with computers".

belief: any QC1 is a QC2.
proof: see, e.g.,
rdan–Lee–Preskill
um algorithms for
n field theories".

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note c

Apparen
Current
from D-V
can be r
simulate

er type 2 (QC2):
universe;
es the
physics
uracy as desired.

l concept of
rs introduced
"Simulating
outers".

y QC1 is a QC2.
e.g.,
Preskill
nms for
ories".

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific
Current "quantum
from D-Wave are
can be more cost-
simulated by tradi

(QC2):

esired.

of

ced

ng

QC2.

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

<u>A note on D-Wave</u>

Apparent scientific consensu
Current "quantum compute
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CP

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;

Quantum computer type 3 (QC3):
efficiently computes anything
that any possible physical
computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must
follow the laws of quantum
physics, so a QC2 can efficiently
simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3. Argument for belief: any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1. Argument for belief: look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Quantum computer type 3 (QC3): efficiently computes anything that any possible physical computer can compute efficiently.

General belief: any QC2 is a QC3.
Argument for belief:
any physical computer must follow the laws of quantum physics, so a QC2 can efficiently simulate any physical computer.

General belief: any QC3 is a QC1.
Argument for belief:
look, we're building a QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

m computer type 3 (QC3):
y computes anything
possible physical
er can compute efficiently.

belief: any QC2 is a QC3.
nt for belief:
sical computer must
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.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The stat

Data ("s
a list of
e.g.: (0,

er type 3 (QC3):
es anything
physical
npute efficiently.

y QC2 is a QC3.
ef:
outer must
 quantum
 can efficiently
ical computer.

y QC3 is a QC1.
ef:
g a QC1.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a cor

Data ("state") sto
a list of 3 element
e.g.: $(0, 0, 0)$.

(QC3):

g

iently.

QC3.

ently

uter.

QC1.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 b
a list of 3 elements of $\{0, 1\}$
e.g.: $(0, 0, 0)$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless—can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.

A note on D-Wave

Apparent scientific consensus: Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits: a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers"
from D-Wave are useless—
can be more cost-effectively
simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is

- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.
e.g.: $(0,1,1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0,1\}$.

A note on D-Wave

Apparent scientific consensus:
Current "quantum computers" from D-Wave are useless— can be more cost-effectively simulated by traditional CPUs.

But D-Wave is
- collecting venture capital;
- selling some machines;
- collecting possibly useful engineering expertise;
- not being punished for deceiving people.

Is D-Wave a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.
e.g.: $(0,1,1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0,1\}$.
e.g.: $(1,1,1,1,1,0,0,0,1,$
$0,0,0,0,0,0,1,1,0,0,0,$
$0,1,0,0,1,0,0,0,0,0,1,$
$1,0,1,0,0,0,1,0,0,0,1,$
$0,0,1,1,1,0,0,1,0,0,0,$
$1,1,0,1,1,0,0,1,0,0,1)$.

on D-Wave

t scientific consensus:
"quantum computers"
Wave are useless—
more cost-effectively
d by traditional CPUs.

Vave is
ting venture capital;
some machines;
ting possibly useful
eering expertise;
eing punished
ceiving people.

ve a bad investment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The stat

Data sto
a list of
e.g.: (3,

e

c consensus:
n computers"
useless—
effectively
tional CPUs.

re capital;
chines;
oly useful
ertise;
ned
ople.

nvestment?

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a qua

Data stored in 3 q
a list of 8 number
e.g.: $(3, 1, 4, 1, 5,$

s:

rs"

Us.

t?

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum com

Data stored in 3 qubits:

a list of 8 numbers, not all z

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

The state of a computer

Data ("state") stored in 3 bits:

a list of 3 elements of $\{0, 1\}$.

e.g.: $(0, 0, 0)$.

e.g.: $(1, 1, 1)$.

e.g.: $(0, 1, 1)$.

Data stored in 64 bits:

a list of 64 elements of $\{0, 1\}$.

e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:

a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of
16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0, 1\}$.
e.g.: $(0, 0, 0)$.
e.g.: $(1, 1, 1)$.
e.g.: $(0, 1, 1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0, 1\}$.
e.g.: $(1, 1, 1, 1, 1, 0, 0, 0, 1,$
$0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0,$
$0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 1,$
$1, 0, 1, 0, 0, 0, 1, 0, 0, 0, 1,$
$0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,$
$1, 1, 0, 1, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

The state of a computer

Data ("state") stored in 3 bits:
a list of 3 elements of $\{0,1\}$.
e.g.: $(0,0,0)$.
e.g.: $(1,1,1)$.
e.g.: $(0,1,1)$.

Data stored in 64 bits:
a list of 64 elements of $\{0,1\}$.
e.g.: $(1,1,1,1,1,0,0,0,1,$
$0,0,0,0,0,0,1,1,0,0,0,$
$0,1,0,0,1,0,0,0,0,0,1,$
$1,0,1,0,0,0,1,0,0,0,1,$
$0,0,1,1,1,0,0,1,0,0,0,$
$1,1,0,1,1,0,0,1,0,0,1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3,1,4,1,5,9,2,6)$.
e.g.: $(-2,7,-1,8,1,-8,-2,8)$.
e.g.: $(0,0,0,0,0,1,0,0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

te of a computer

state") stored in 3 bits:

3 elements of $\{0, 1\}$.

$0, 0)$.

$1, 1)$.

$1, 1)$.

ored in 64 bits:

64 elements of $\{0, 1\}$.

$1, 1, 1, 1, 0, 0, 0, 1,$

$, 0, 0, 1, 1, 0, 0, 0,$

$, 1, 0, 0, 0, 0, 0, 1,$

$, 0, 0, 1, 0, 0, 0, 1,$

$, 1, 0, 0, 1, 0, 0, 0,$

$, 1, 0, 0, 1, 0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.

e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.

e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.

e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: $(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measurin

Can sim

Cannot

of numb

mputer

ored in 3 bits:

s of $\{0, 1\}$.

bits:

nts of $\{0, 1\}$.

$0, 0, 0, 1,$

$0, 0, 0,$

$0, 0, 1,$

$0, 0, 1,$

$0, 0, 0,$

$0, 0, 1)$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: $(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quant

Can simply look a

Cannot simply loo

of numbers stored

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.: $(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum comp

Can simply look at a bit.
Cannot simply look at the li
of numbers stored in n qubi

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

The state of a quantum computer

Data stored in 3 qubits:
a list of 8 numbers, not all zero.
e.g.: $(3, 1, 4, 1, 5, 9, 2, 6)$.
e.g.: $(-2, 7, -1, 8, 1, -8, -2, 8)$.
e.g.: $(0, 0, 0, 0, 0, 1, 0, 0)$.

Data stored in 4 qubits: a list of 16 numbers, not all zero. e.g.:
$(3, 1, 4, 1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

Data stored in 64 qubits:
a list of 2^{64} numbers, not all zero.

Data stored in 1000 qubits: a list of 2^{1000} numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

te of a quantum computer

ored in 3 qubits:

8 numbers, not all zero.

$1, 4, 1, 5, 9, 2, 6)$.

$2, 7, -1, 8, 1, -8, -2, 8)$.

$0, 0, 0, 0, 1, 0, 0)$.

ored in 4 qubits: a list of

bers, not all zero. e.g.:

$1, 5, 9, 2, 6, 5, 3, 5, 8, 9, 7, 9, 3)$.

ored in 64 qubits:

2^{64} numbers, not all zero.

ored in 1000 qubits: a list

numbers, not all zero.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list
of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state
$(a_0, a_1, \ldots, a_{2^n-1})$ then
measurement produces q
with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros
except 1 at position q.

e.g.: Say

$(1, 1, 1,$

antum computer

ubits:

s, not all zero.

$9, 2, 6)$.

$, 1, -8, -2, 8)$.

$1, 0, 0)$.

ubits: a list of

ll zero. e.g.:

$, 5, 3, 5, 8, 9, 7, 9, 3)$.

qubits:

ers, not all zero.

00 qubits: a list

not all zero.

Measuring a quantum computer

Can simply look at a bit.

Cannot simply look at the list

of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state

$(a_0, a_1, \ldots, a_{2^n-1})$ then

measurement produces q

with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros

except 1 at position q.

e.g.: Say 3 qubits

$(1, 1, 1, 1, 1, 1, 1, 1$

nputer

zero.

2, 8).

st of
g.:
9, 7, 9, 3).

l zero.

a list
ro.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have stat
$(1, 1, 1, 1, 1, 1, 1, 1)$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1,1,1,1,1,1,1,1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Measuring a quantum computer

Can simply look at a bit.
Cannot simply look at the list of numbers stored in n qubits.

Measuring n qubits

- produces n bits and
- destroys the state.

If n qubits have state $(a_0, a_1, \ldots, a_{2^n-1})$ then measurement produces q with probability $|a_q|^2 / \sum_r |a_r|^2$.

State is then all zeros except 1 at position q.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

ng a quantum computer

ply look at a bit.
simply look at the list
ers stored in n qubits.

ing n qubits
ces n bits and
ys the state.

its have state
$\ldots, a_{2^n-1})$ then
ment produces q
bability $|a_q|^2/\sum_r |a_r|^2$.

then all zeros
at position q.

e.g.: Say 3 qubits have state
$(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold
today are measurably biased.

e.g.: Say
$(3, 1, 4, 1$

tum computer

t a bit.

k at the list

in n qubits.

its

and

te.

ate

then

luces q

$_q|^2/\sum_r |a_r|^2$.

eros

on q.

e.g.: Say 3 qubits have state $(1,1,1,1,1,1,1,1)$.

Measurement produces

$000 = 0$ with probability $1/8$;

$001 = 1$ with probability $1/8$;

$010 = 2$ with probability $1/8$;

$011 = 3$ with probability $1/8$;

$100 = 4$ with probability $1/8$;

$101 = 5$ with probability $1/8$;

$110 = 6$ with probability $1/8$;

$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits

$(3,1,4,1,5,9,2,6$

uter

st

ts.

$a_r|^2$.

e.g.: Say 3 qubits have state $(1, 1, 1, 1, 1, 1, 1, 1)$.

Measurement produces
$000 = 0$ with probability $1/8$;
$001 = 1$ with probability $1/8$;
$010 = 2$ with probability $1/8$;
$011 = 3$ with probability $1/8$;
$100 = 4$ with probability $1/8$;
$101 = 5$ with probability $1/8$;
$110 = 6$ with probability $1/8$;
$111 = 7$ with probability $1/8$.

"Quantum RNG."

Warning: Quantum RNGs sold today are measurably biased.

e.g.: Say 3 qubits have 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
$1, 1, 1, 1, 1)$.

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.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

e.g.: Say 3 qubits have state $(3, 1, 4, 1, 5, 9, 2, 6)$.

Measurement produces
$000 = 0$ with probability $9/173$;
$001 = 1$ with probability $1/173$;
$010 = 2$ with probability $16/173$;
$011 = 3$ with probability $1/173$;
$100 = 4$ with probability $25/173$;
$101 = 5$ with probability $81/173$;
$110 = 6$ with probability $4/173$;
$111 = 7$ with probability $36/173$.

5 is most likely outcome.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

y 3 qubits have state

$1, 5, 9, 2, 6)$.

ement produces

with probability $9/173$;

with probability $1/173$;

with probability $16/173$;

with probability $1/173$;

with probability $25/173$;

with probability $81/173$;

with probability $4/173$;

with probability $36/173$.

t likely outcome.

e.g.: Say 3 qubits have state

$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

$000 = 0$ with probability 0;

$001 = 1$ with probability 0;

$010 = 2$ with probability 0;

$011 = 3$ with probability 0;

$100 = 4$ with probability 0;

$101 = 5$ with probability 1;

$110 = 6$ with probability 0;

$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT ga

NOT_0 g

$(3, 1, 4, 1$

$(1, 3, 1, 4$

have state

).

duces

ability $9/173$;

ability $1/173$;

ability $16/173$;

ability $1/173$;

ability $25/173$;

ability $81/173$;

ability $4/173$;

ability $36/173$.

tcome.

e.g.: Say 3 qubits have state
$(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces

$000 = 0$ with probability 0;

$001 = 1$ with probability 0;

$010 = 2$ with probability 0;

$011 = 3$ with probability 0;

$100 = 4$ with probability 0;

$101 = 5$ with probability 1;

$110 = 6$ with probability 0;

$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 q

$(3, 1, 4, 1, 5, 9, 2, 6$

$(1, 3, 1, 4, 9, 5, 6, 2$

e

173;
173;
/173;
173;
/173;
/173;
173;
/173.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.

e.g.: Say 3 qubits have state $(0, 0, 0, 0, 0, 1, 0, 0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto (1, 3, 1, 4, 9, 5, 6, 2)$.

e.g.: Say 3 qubits have state $(0,0,0,0,0,1,0,0)$.

Measurement produces
$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

NOT gates

NOT_0 gate on 3 qubits:
$(3,1,4,1,5,9,2,6) \mapsto (1,3,1,4,9,5,6,2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto (1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

e.g.: Say 3 qubits have state $(0,0,0,0,0,1,0,0)$.

Measurement produces

$000 = 0$ with probability 0;
$001 = 1$ with probability 0;
$010 = 2$ with probability 0;
$011 = 3$ with probability 0;
$100 = 4$ with probability 0;
$101 = 5$ with probability 1;
$110 = 6$ with probability 0;
$111 = 7$ with probability 0.

5 is guaranteed outcome.

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, 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 measuremer
flipping bit 0 of result.
Flip: output is not input.

NOT gates

NOT_0 gate on 3 qubits:
$(3,1,4,1,5,9,2,6) \mapsto$
$(1,3,1,4,9,5,6,2)$.

NOT_0 gate on 4 qubits:
$(3,1,4,1,5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$
$(1,3,1,4,9,5,6,2,3,5,8,5,7,9,3,9)$.

NOT_1 gate on 3 qubits:
$(3,1,4,1,5,9,2,6) \mapsto$
$(4,1,3,1,2,6,5,9)$.

NOT_2 gate on 3 qubits:
$(3,1,4,1,5,9,2,6) \mapsto$
$(5,9,2,6,3,1,4,1)$.

state	measurement
$(1,0,0,0,0,0,0,0)$	000
$(0,1,0,0,0,0,0,0)$	001
$(0,0,1,0,0,0,0,0)$	010
$(0,0,0,1,0,0,0,0)$	011
$(0,0,0,0,1,0,0,0)$	100
$(0,0,0,0,0,1,0,0)$	101
$(0,0,0,0,0,0,1,0)$	110
$(0,0,0,0,0,0,0,1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

tes

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$4, 9, 5, 6, 2).$

ate on 4 qubits:

$5,9,2,6,5,3,5,8,9,7,9,3) \mapsto$

$9,5,6,2,3,5,8,5,7,9,3,9).$

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$1, 2, 6, 5, 9).$

ate on 3 qubits:

$1, 5, 9, 2, 6) \mapsto$

$6, 3, 1, 4, 1).$

state	measurement
$(1, 0, 0, 0, 0, 0, 0, 0)$	000
$(0, 1, 0, 0, 0, 0, 0, 0)$	001
$(0, 0, 1, 0, 0, 0, 0, 0)$	010
$(0, 0, 0, 1, 0, 0, 0, 0)$	011
$(0, 0, 0, 0, 1, 0, 0, 0)$	100
$(0, 0, 0, 0, 0, 1, 0, 0)$	101
$(0, 0, 0, 0, 0, 0, 1, 0)$	110
$(0, 0, 0, 0, 0, 0, 0, 1)$	111

Operation on quantum state:
NOT_0, swapping pairs.
Operation after measurement:
flipping bit 0 of result.
Flip: output is not input.

Controll

e.g. CN

$(3, 1, 4, 1$

$(3, 1, 1, 4$

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 g

e.g. $\text{CNOT}_{1,0}$:

$(3,1,4,1,5,9,2,6$

$(3,1,1,4,5,9,6,2$

$,3) \mapsto$
$,9).$

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 gates

e.g. $\text{CNOT}_{1,0}$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

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 gates

e.g. $\text{CNOT}_{1,0}$:

$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

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 gates

e.g. $\mathrm{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

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 gates

e.g. $\text{CNOT}_{1,0}$:

$(3,1,4,1,5,9,2,6) \mapsto$

$(3,1,1,4,5,9,6,2)$.

Operation after measurement:

flipping bit 0 *if* bit 1 is set; i.e.,

$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:

$(3,1,4,1,5,9,2,6) \mapsto$

$(3,1,4,1,9,5,6,2)$.

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 gates

e.g. $\text{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\text{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

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 gates

e.g. $\text{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\text{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli g

Also kno
controlle

e.g. CCN
(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 gates

e.g. $\mathrm{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\mathrm{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\mathrm{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as
controlled-controll

e.g. $\mathrm{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6$
$(3, 1, 4, 1, 5, 9, 6, 2$

ement

e:

nt:

Controlled-NOT gates

e.g. $\mathrm{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\mathrm{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\mathrm{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as
controlled-controlled-NOT g

e.g. $\mathrm{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Controlled-NOT gates

e.g. $\text{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\text{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Controlled-NOT gates

e.g. $\mathrm{CNOT}_{1,0}$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,1,4,5,9,6,2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2,q_1,q_0) \mapsto (q_2,q_1,q_0 \oplus q_1)$.

e.g. $\mathrm{CNOT}_{2,0}$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,9,5,6,2)$.

e.g. $\mathrm{CNOT}_{0,2}$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,9,4,6,5,1,2,1)$.

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\mathrm{CCNOT}_{2,1,0}$:
$(3,1,4,1,5,9,2,6) \mapsto$
$(3,1,4,1,5,9,6,2)$.

Operation after measurement:
$(q_2,q_1,q_0) \mapsto (q_2,q_1,q_0 \oplus q_1 q_2)$.

Controlled-NOT gates

e.g. $\text{CNOT}_{1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 1, 4, 5, 9, 6, 2)$.

Operation after measurement:
flipping bit 0 *if* bit 1 is set; i.e.,
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

e.g. $\text{CNOT}_{2,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 9, 5, 6, 2)$.

e.g. $\text{CNOT}_{0,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 9, 4, 6, 5, 1, 2, 1)$.

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\text{CCNOT}_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

ed-NOT gates

OT$_{1,0}$:
1, 5, 9, 2, 6) $\mapsto$
4, 5, 9, 6, 2).

on after measurement:
bit 0 *if* bit 1 is set; i.e.,
$q_0) \mapsto (q_2, q_1, q_0 \oplus q_1)$.

OT$_{2,0}$:
1, 5, 9, 2, 6) $\mapsto$
1, 9, 5, 6, 2).

OT$_{0,2}$:
1, 5, 9, 2, 6) $\mapsto$
6, 5, 1, 2, 1).

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. CCNOT$_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. CCNOT$_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More sh

Combine
to build

ates

) $\mapsto$

).

easurement:

t 1 is set; i.e.,

, $q_1, q_0 \oplus q_1$).

) $\mapsto$

).

) $\mapsto$

).

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\text{CCNOT}_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CN
to build other perr

nt:
i.e.,
q_1).

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\text{CCNOT}_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toff
to build other permutations.

Toffoli gates

Also known as
controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\text{CCNOT}_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

Toffoli gates

Also known as controlled-controlled-NOT gates.

e.g. $\text{CCNOT}_{2,1,0}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 1, 5, 9, 6, 2)$.

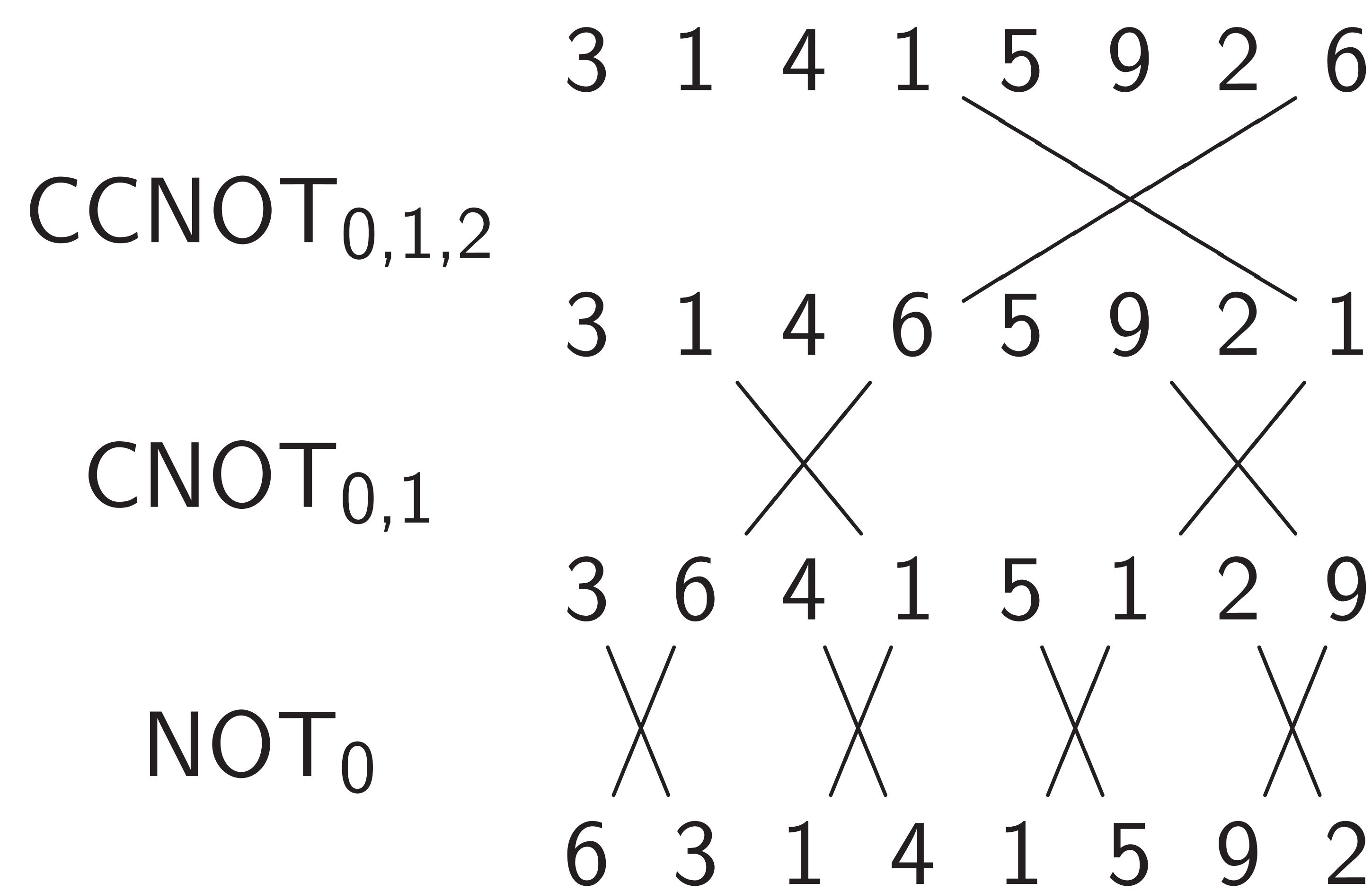
Operation after measurement:
$(q_2, q_1, q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

e.g. $\text{CCNOT}_{0,1,2}$:
$(3, 1, 4, 1, 5, 9, 2, 6) \mapsto$
$(3, 1, 4, 6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

e.g. series of gates to rotate 8 positions by distance 1:



ates

own as
ed-controlled-NOT gates.

NOT$_{2,1,0}$:
$1, 5, 9, 2, 6) \mapsto$
$1, 5, 9, 6, 2)$.

on after measurement:
$q_0) \mapsto (q_2, q_1, q_0 \oplus q_1 q_2)$.

NOT$_{0,1,2}$:
$1, 5, 9, 2, 6) \mapsto$
$6, 5, 9, 2, 1)$.

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

e.g. series of gates to
rotate 8 positions by distance 1:

3 1 4 1 5 9 2 6

CCNOT$_{0,1,2}$

3 1 4 6 5 9 2 1

CNOT$_{0,1}$

3 6 4 1 5 1 2 9

NOT$_0$

6 3 1 4 1 5 9 2

Hadama

Hadama

$(a, b) \mapsto$

3 1

4 2

ed-NOT gates.

) $\mapsto$

).

easurement:

, $q_1, q_0 \oplus q_1 q_2$).

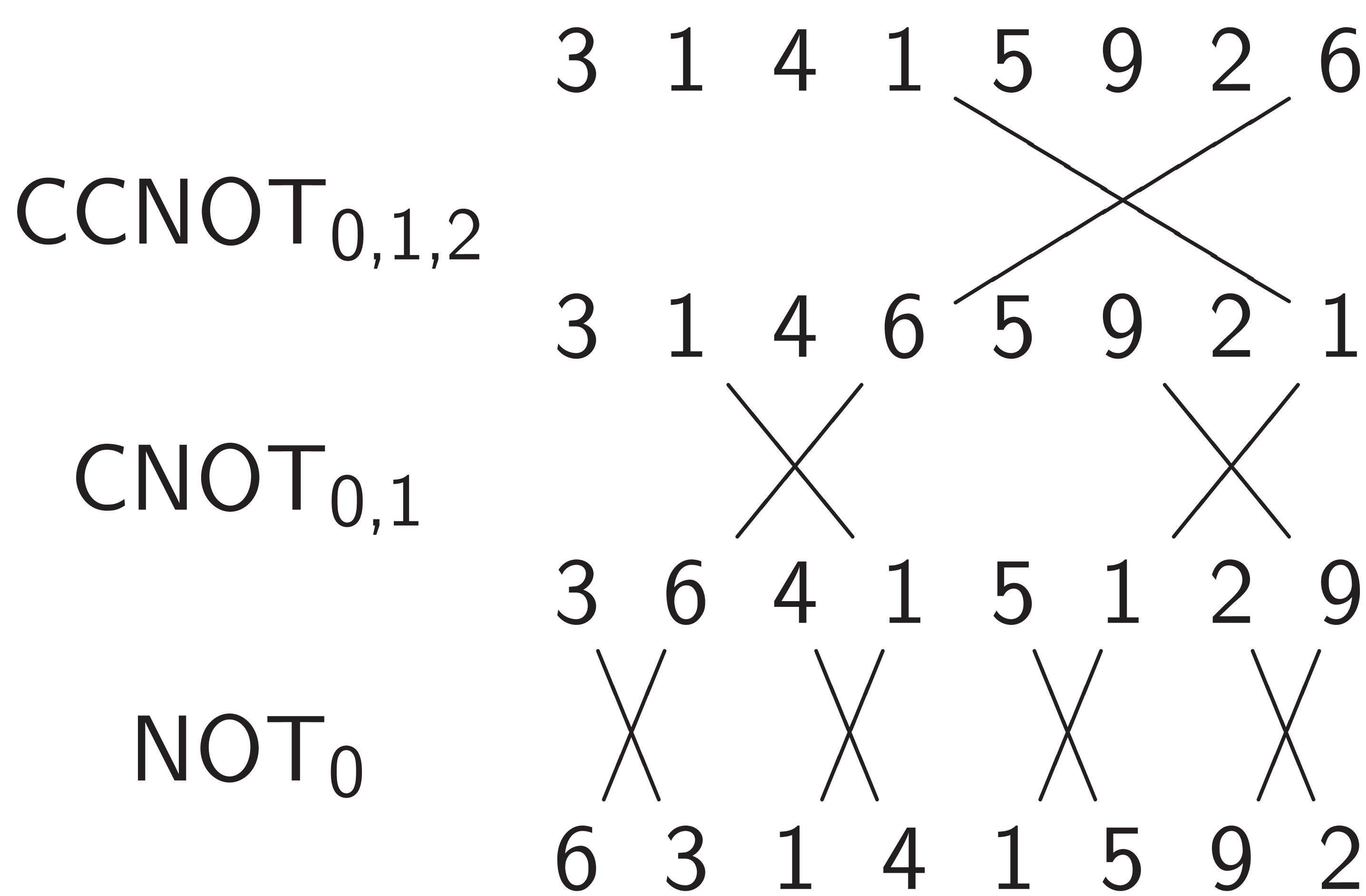
) $\mapsto$

).

More shuffling

Combine NOT, CNOT, Toffoli
to build other permutations.

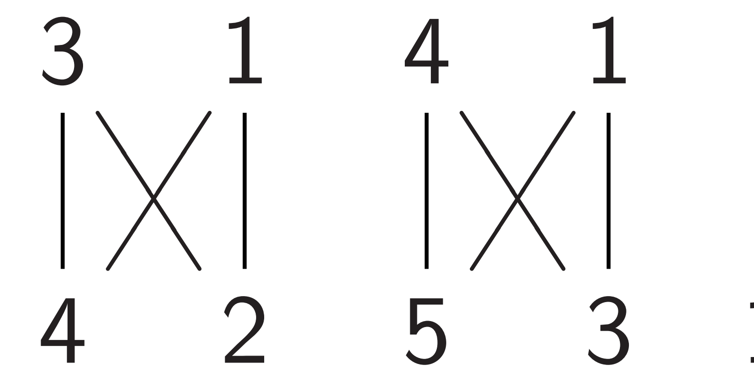
e.g. series of gates to
rotate 8 positions by distance 1:



Hadamard gates

Hadamard$_0$:

$(a, b) \mapsto (a + b, a$



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

$\text{CCNOT}_{0,1,2}$

3 1 4 6 5 9 2 1

$\text{CNOT}_{0,1}$

3 6 4 1 5 1 2 9

NOT_0

6 3 1 4 1 5 9 2

Hadamard gates

Hadamard_0:

$(a, b) \mapsto (a + b, a - b)$.

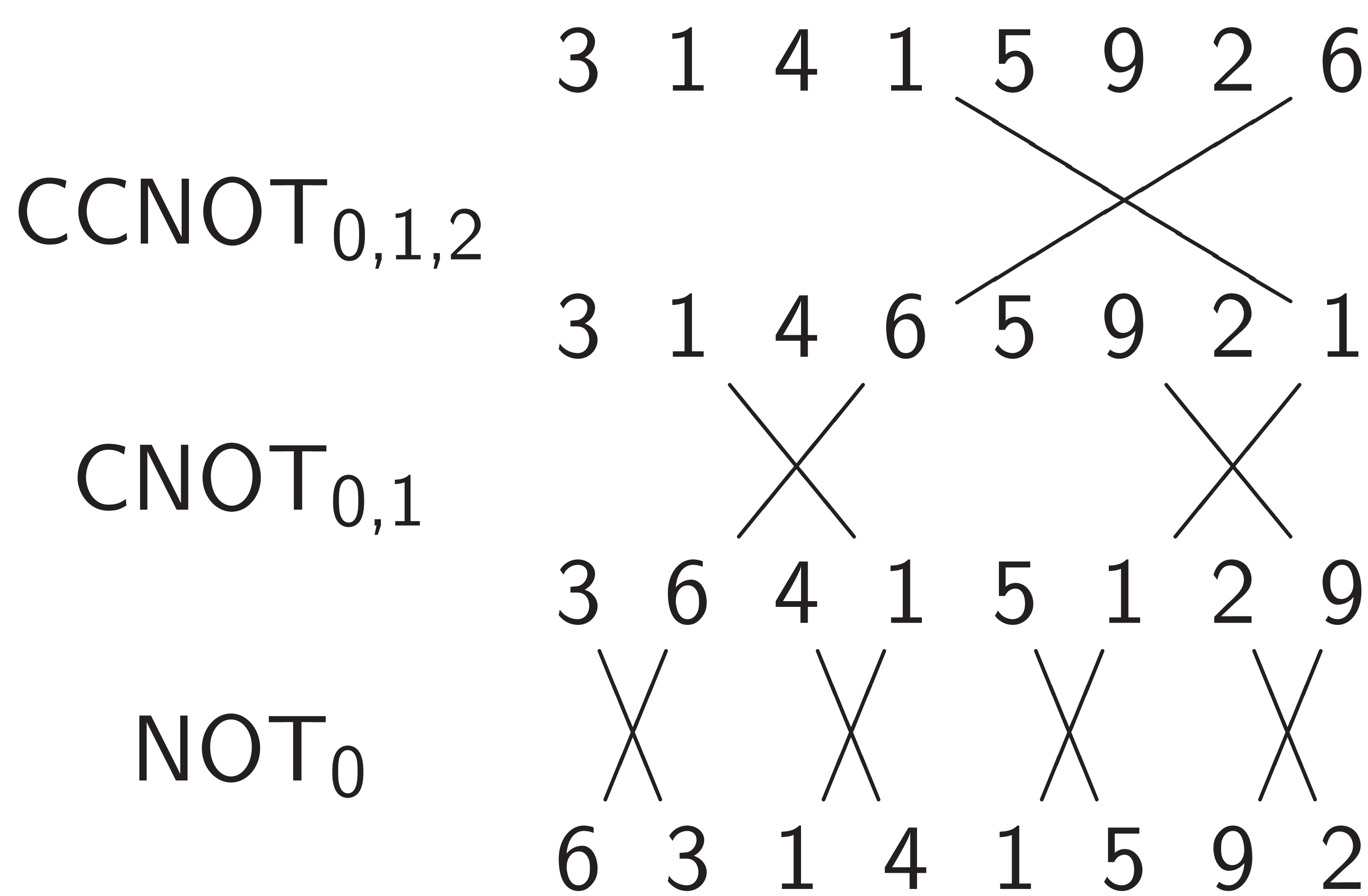
3 1   4 1   5 9   2

4 2   5 3   14 −4   8

More shuffling

Combine NOT, CNOT, Toffoli to build other permutations.

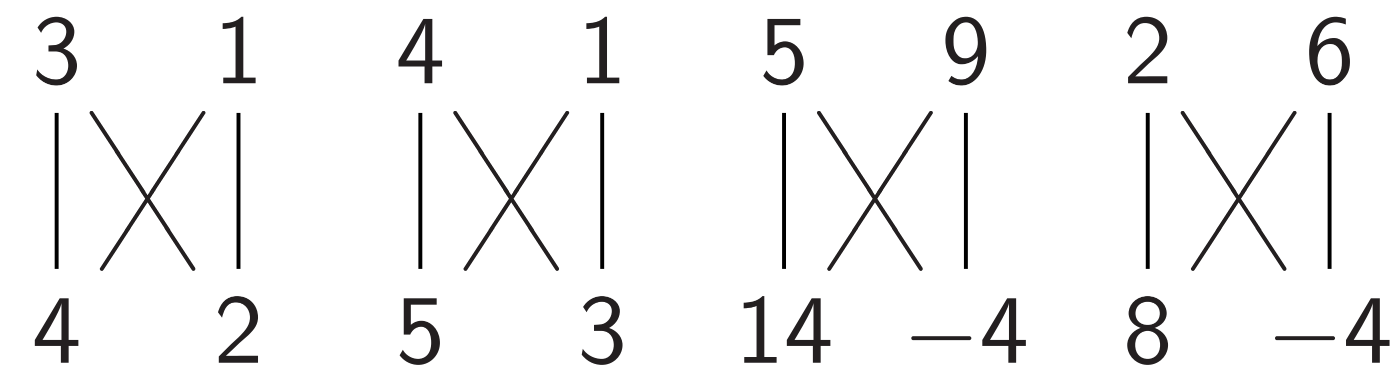
e.g. series of gates to rotate 8 positions by distance 1:



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
$\text{CCNOT}_{0,1,2}$								
	3	1	4	6	5	9	2	1
$\text{CNOT}_{0,1}$								
	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:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.

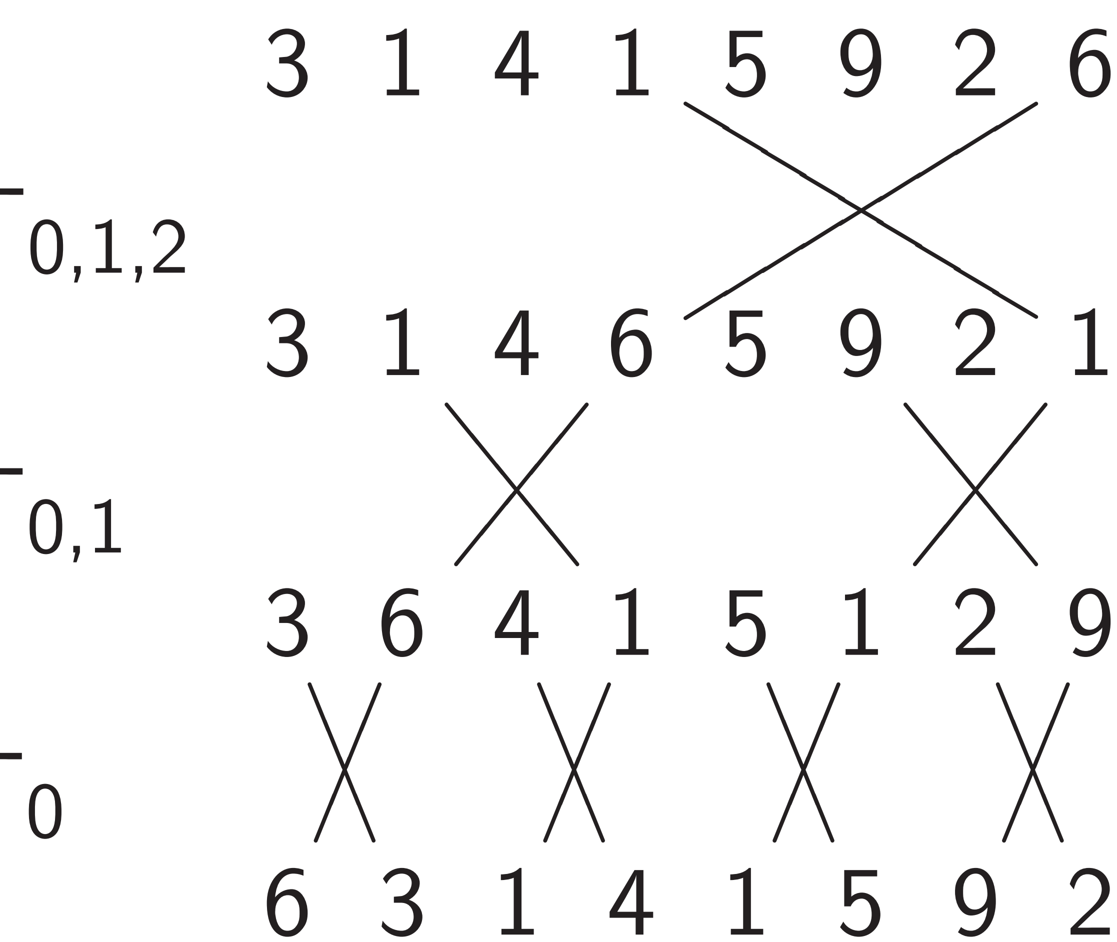
3 1 4 1   5 9 2 6

7 2 −1 0   7 15 3 3

uffling

e NOT, CNOT, Toffoli
other permutations.

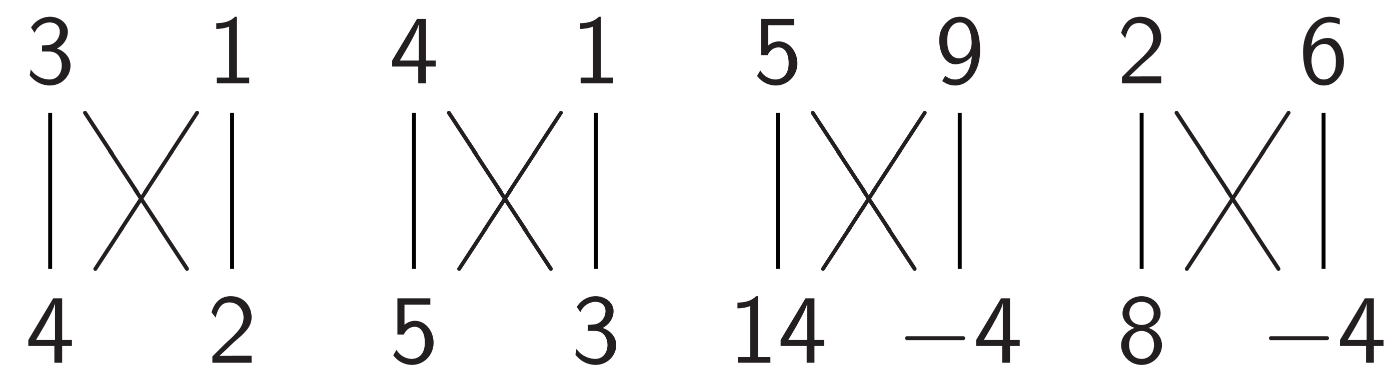
es of gates to
positions by distance 1:



Hadamard gates

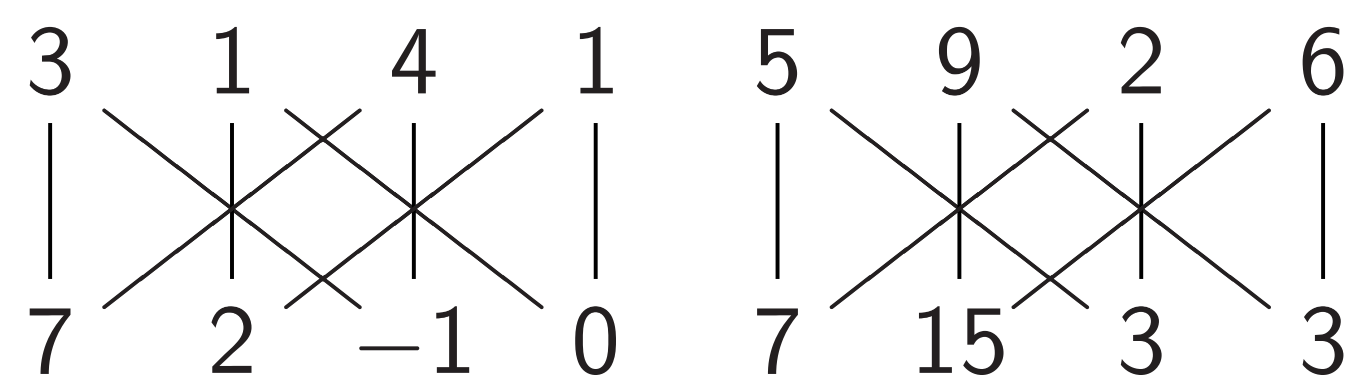
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



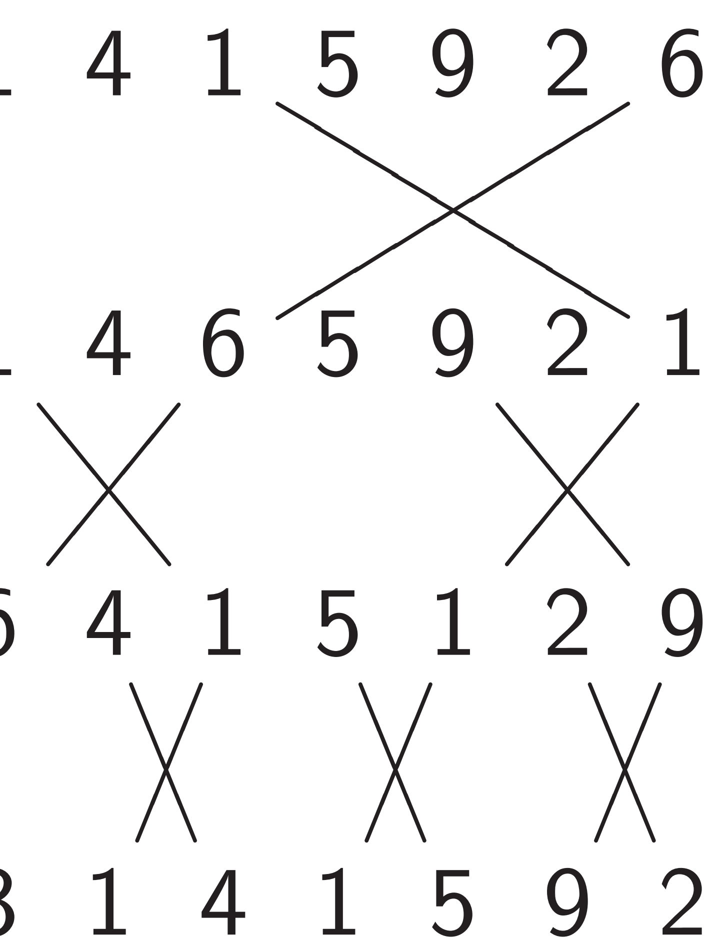
Simon's

Step 1.

1, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0
0, 0, 0, 0

NOT, Toffoli
mutations.

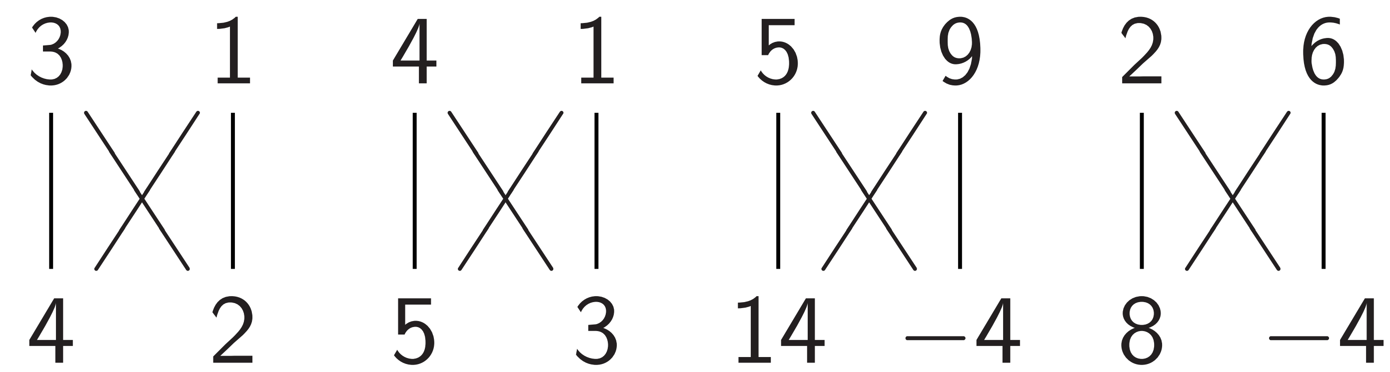
s to
by distance 1:



Hadamard gates

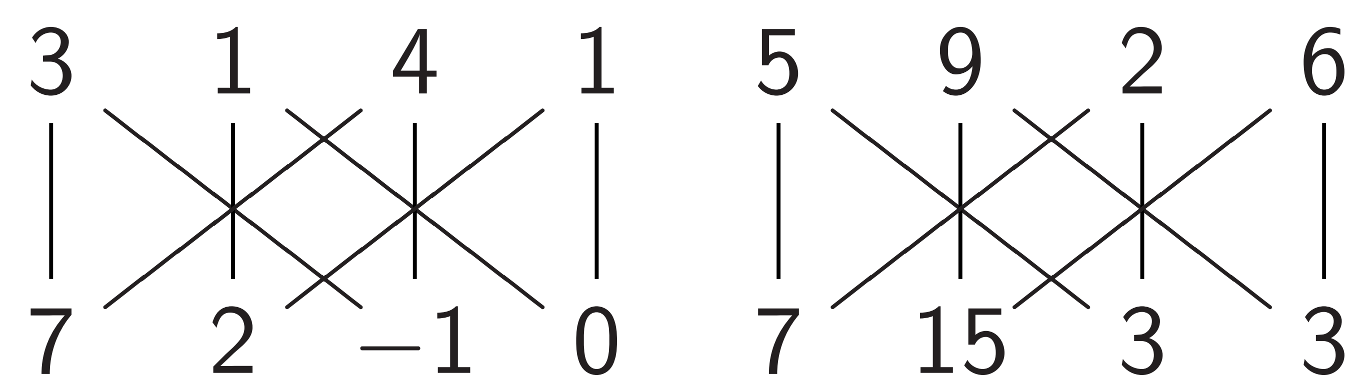
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



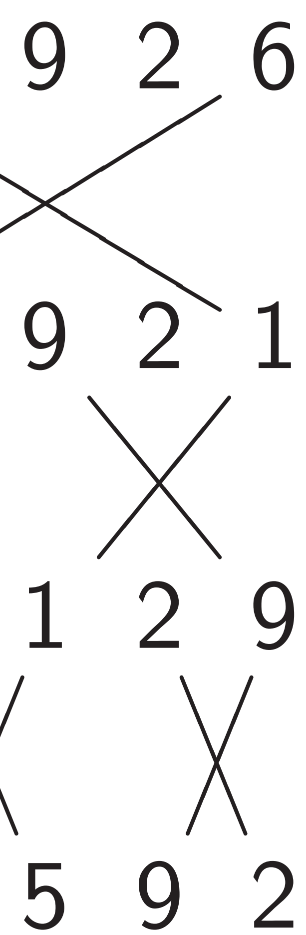
Simon's algorithm

Step 1. Set up pu

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

oli

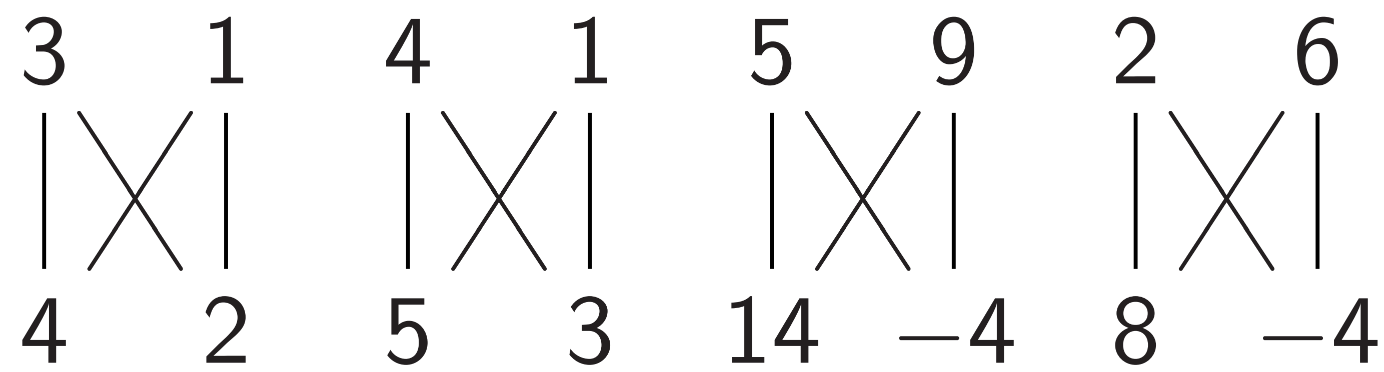
ce 1:



Hadamard gates

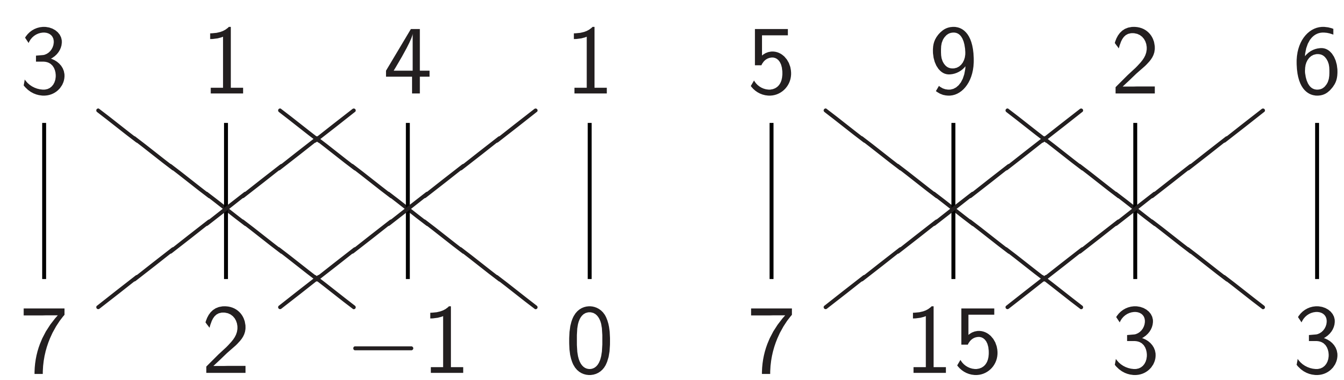
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

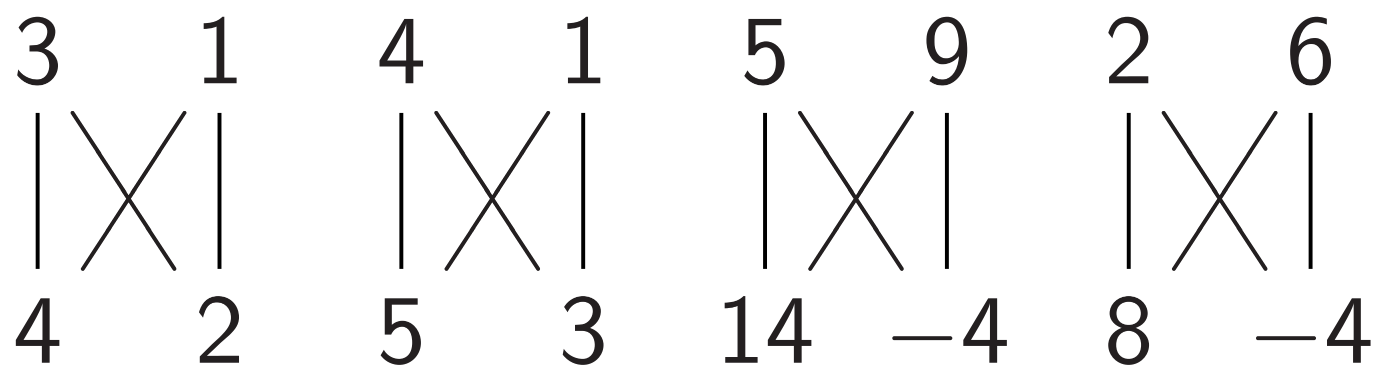
Step 1. Set up pure zero sta

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.

Hadamard gates

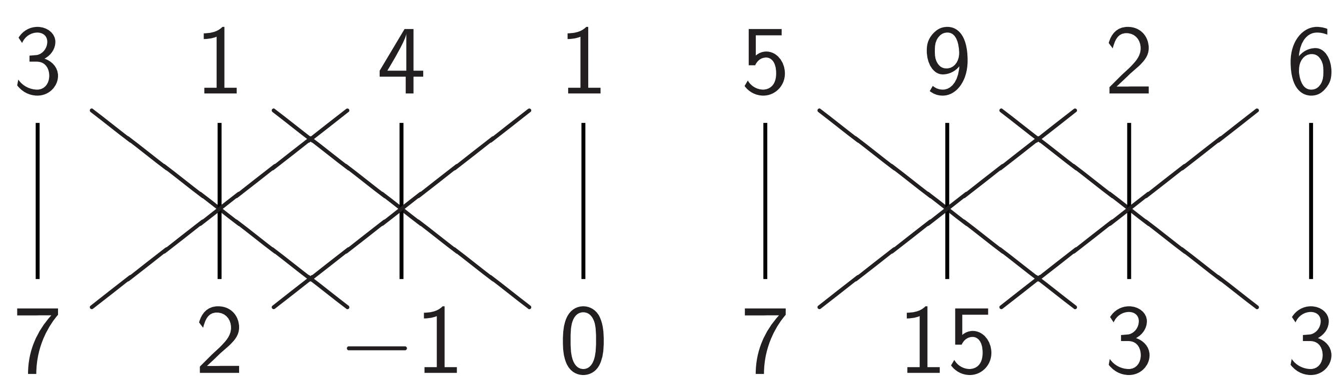
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

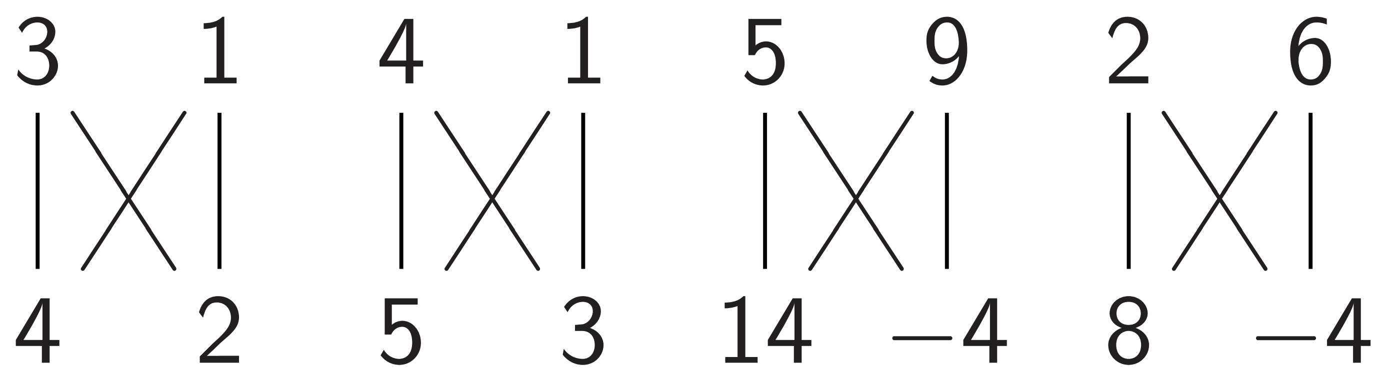
Step 1. Set up pure zero state:

1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Hadamard gates

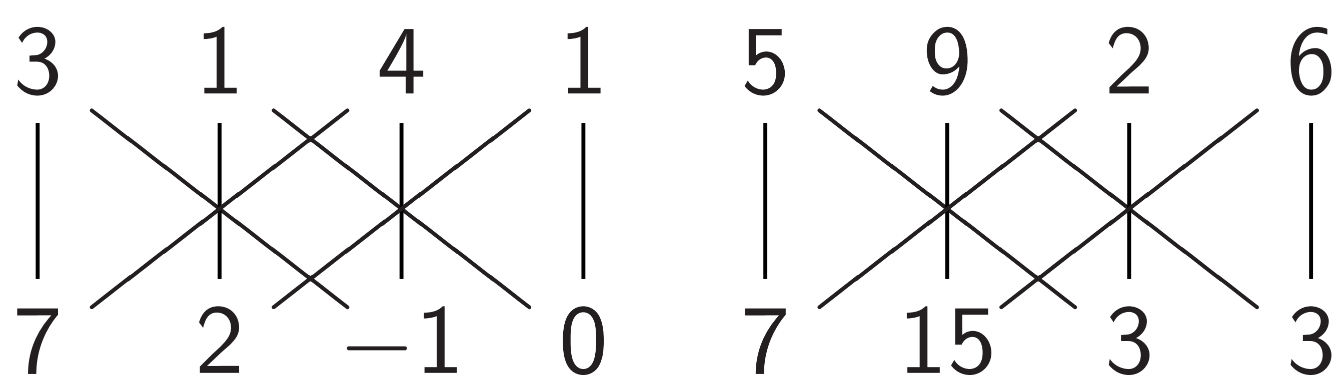
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

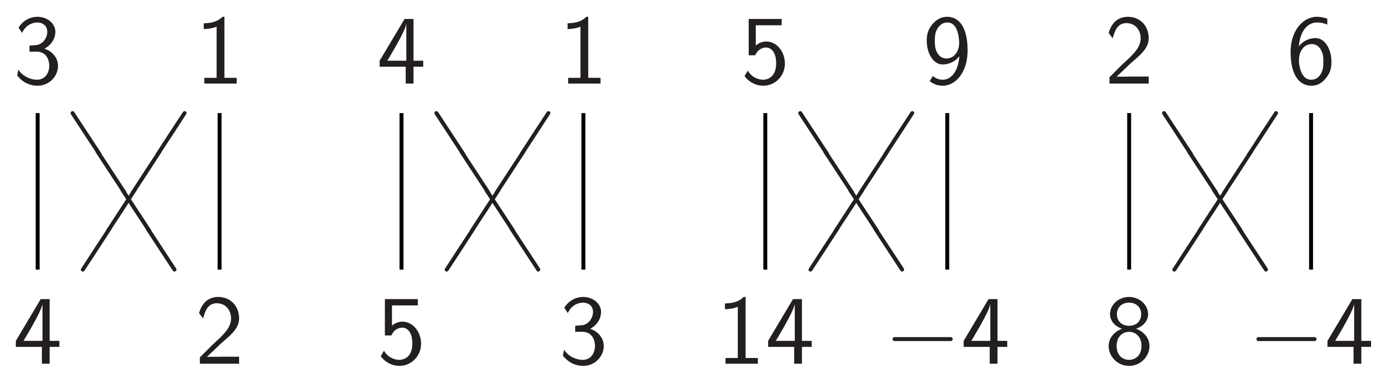
Step 2. Hadamard$_0$:

1, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Hadamard gates

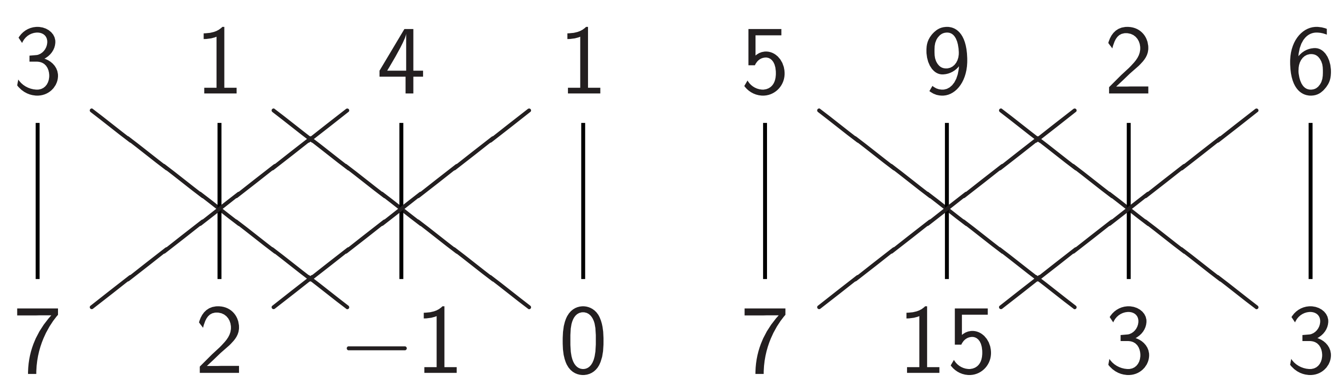
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

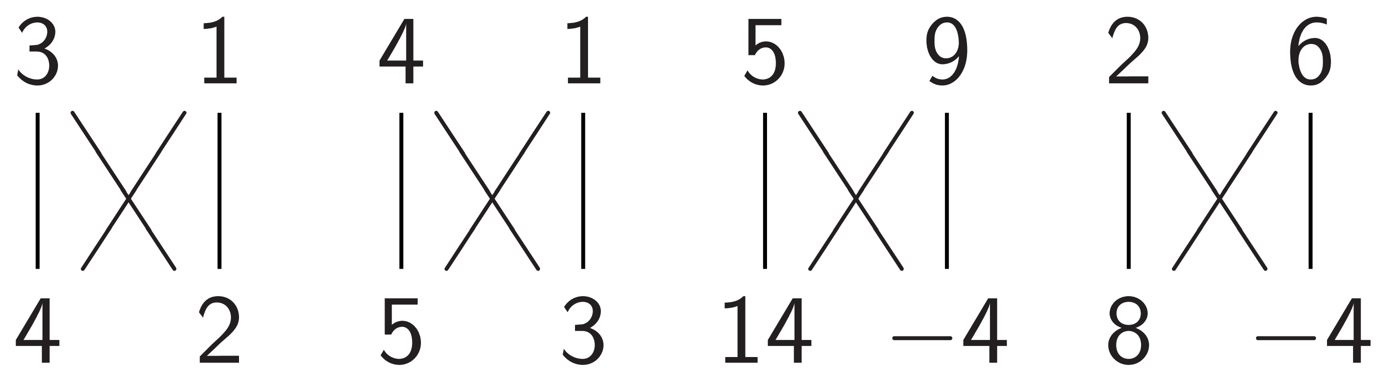
Step 3. Hadamard$_1$:

1, 1, 1, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Hadamard gates

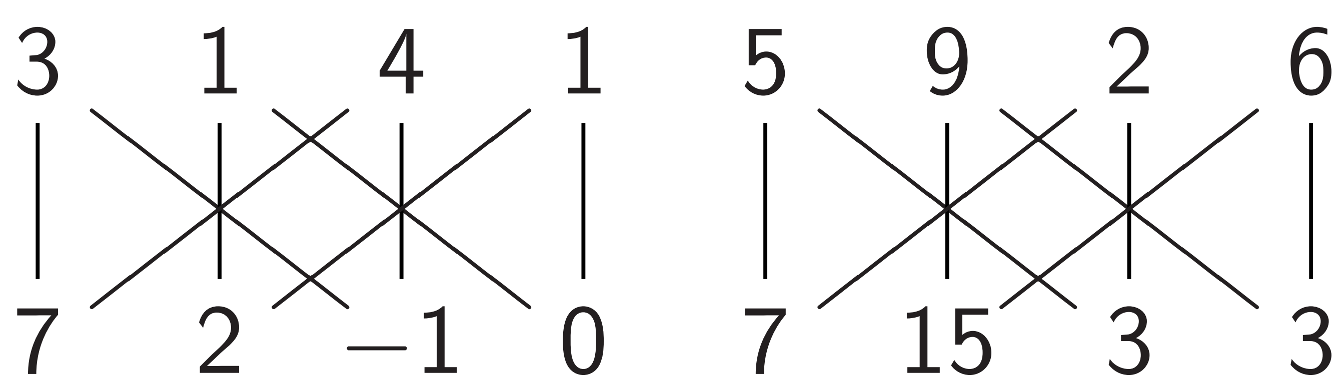
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 4. Hadamard$_2$:

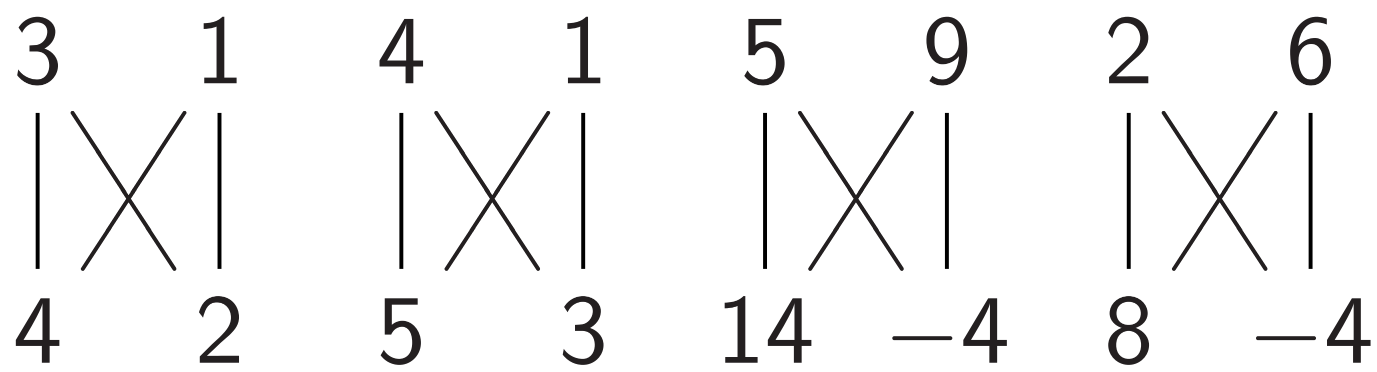
1, 1, 1, 1, 1, 1, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe.

Hadamard gates

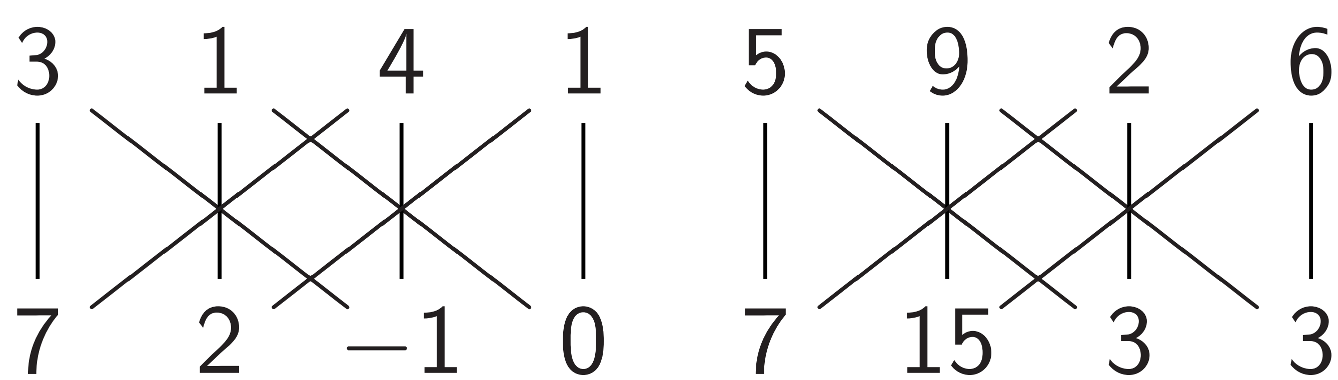
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5. CNOT$_{0,3}$:

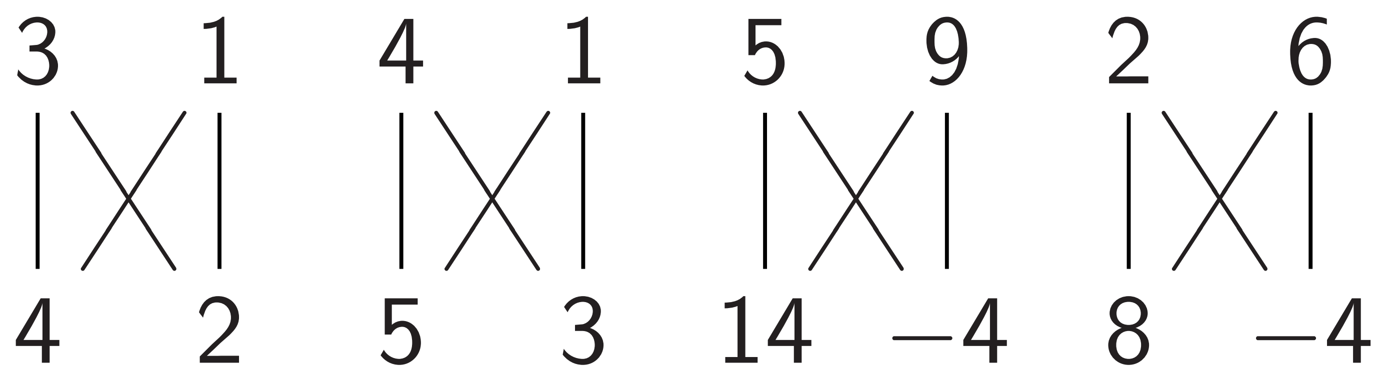
1, 0, 1, 0, 1, 0, 1, 0,
0, 1, 0, 1, 0, 1, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

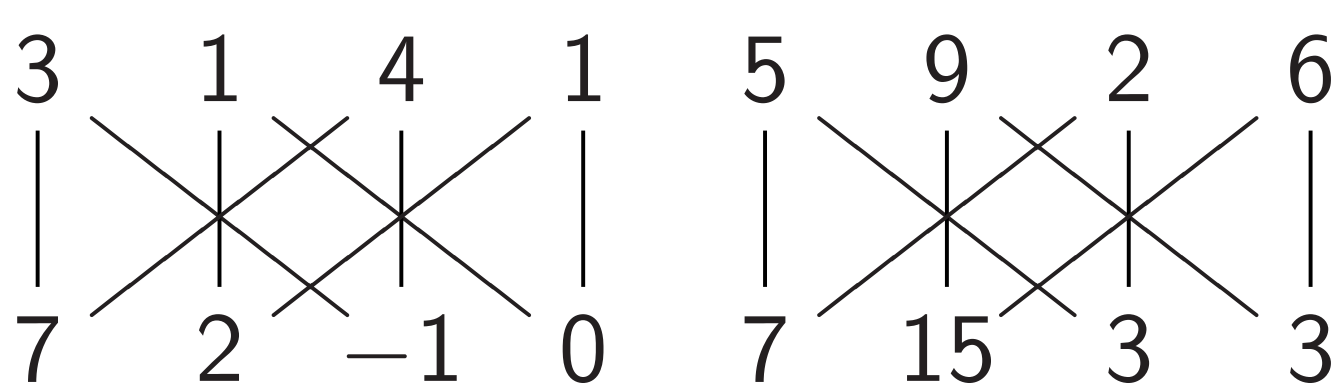
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5b. More shuffling:

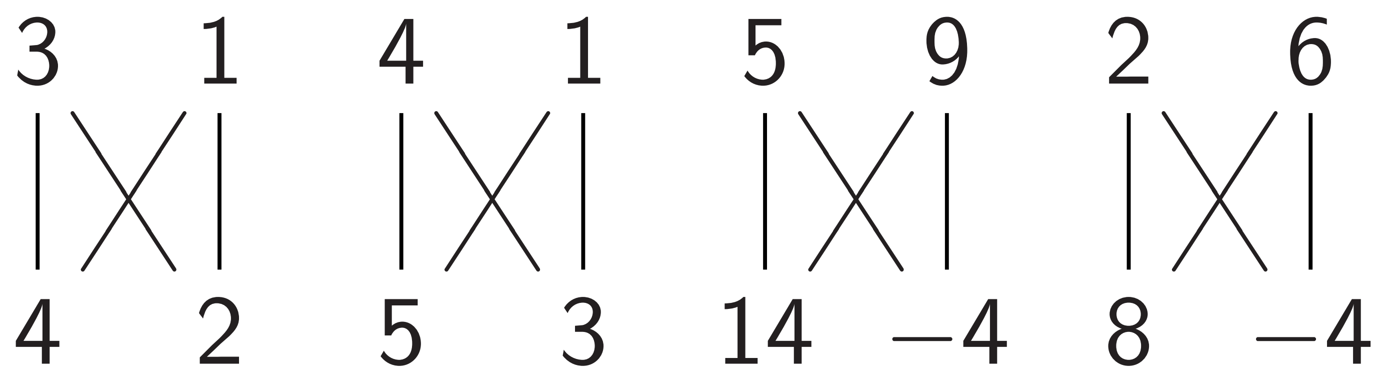
1, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

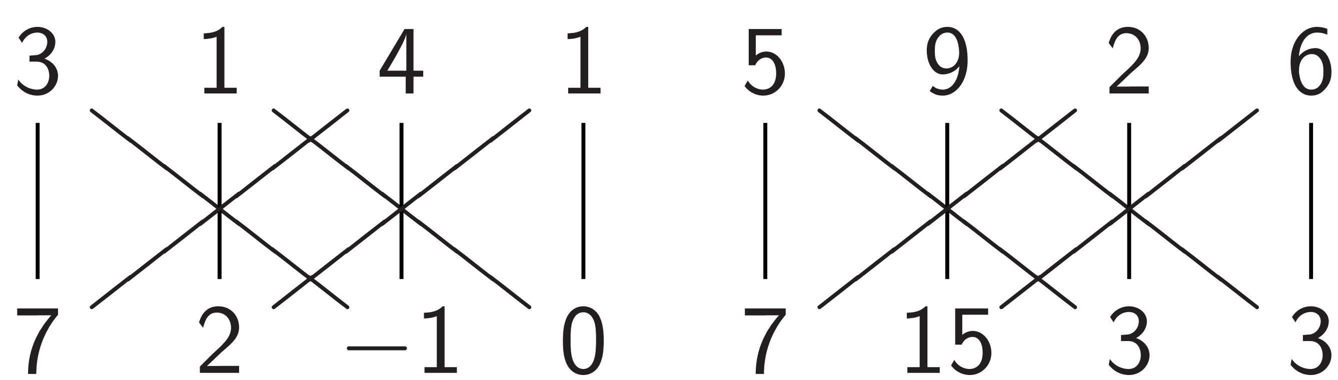
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5c. More shuffling:

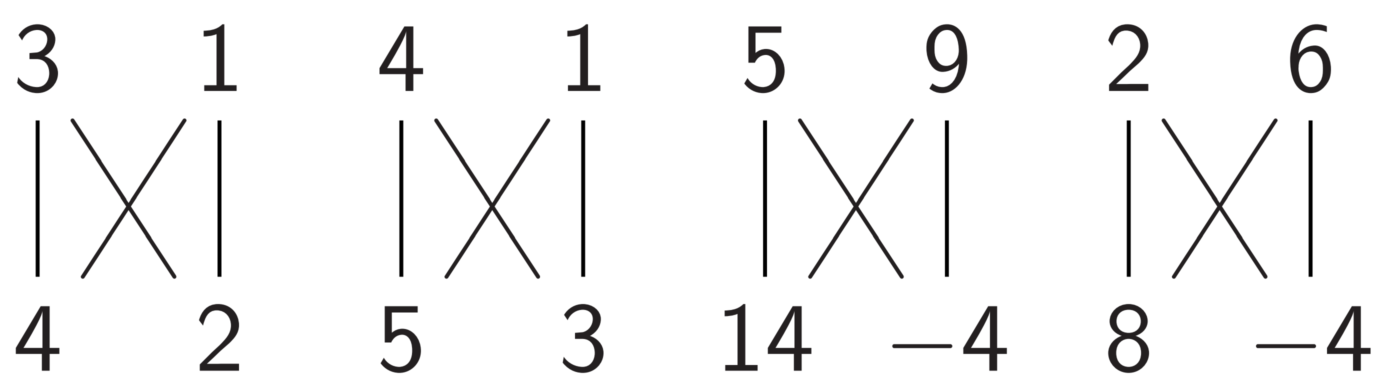
1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 1.

Each column is a parallel universe performing its own computations.

Hadamard gates

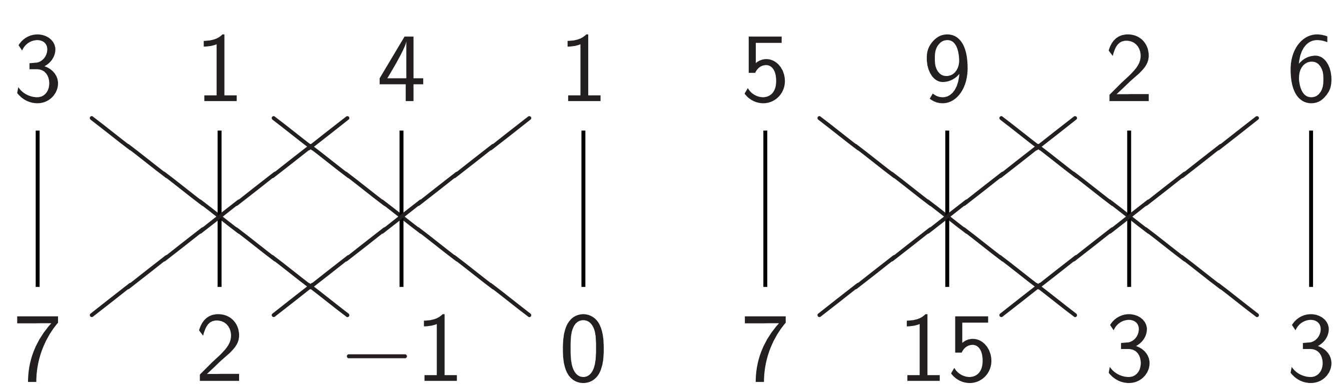
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5d. More shuffling:

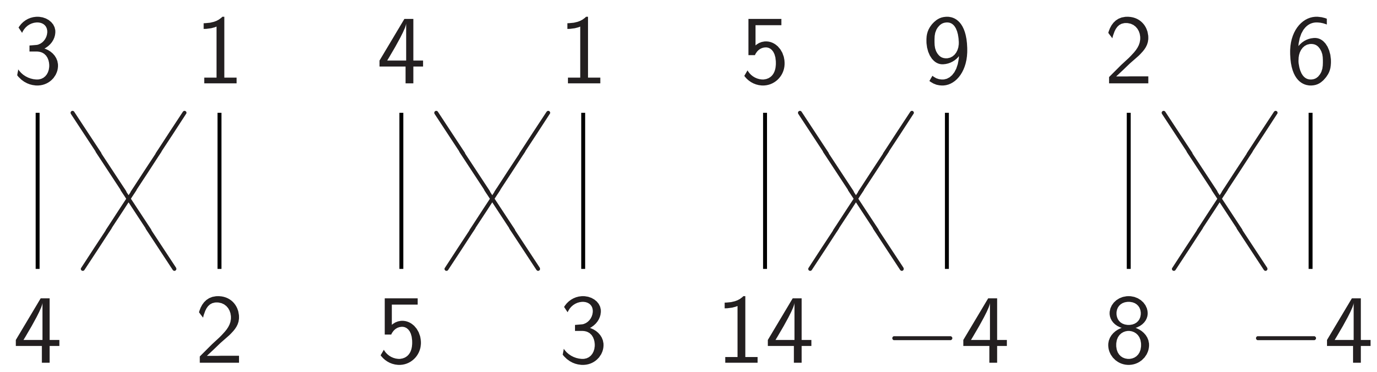
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

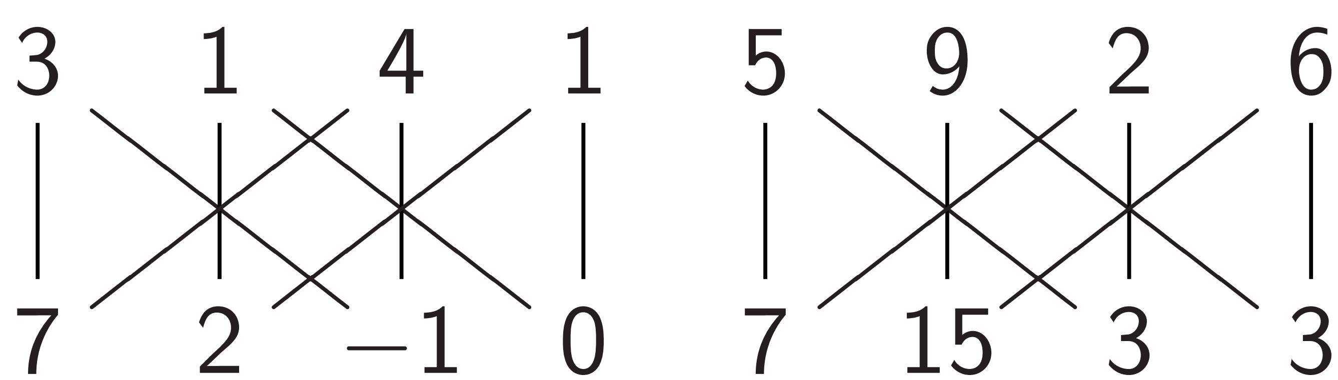
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5e. More shuffling:

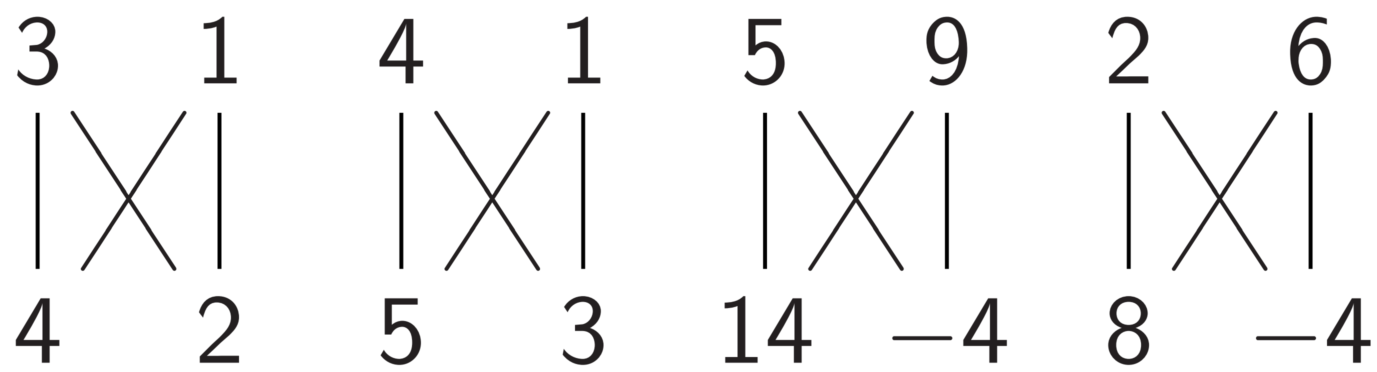
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

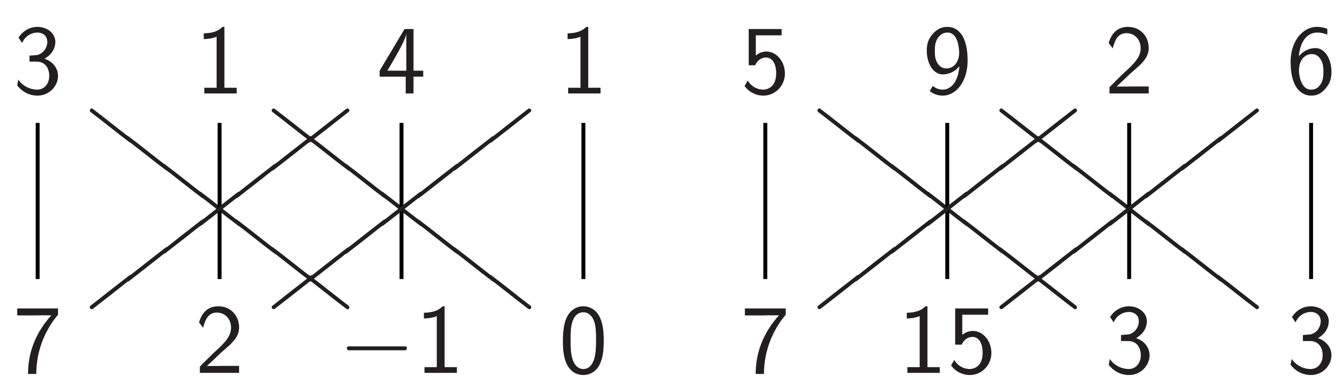
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5f. More shuffling:

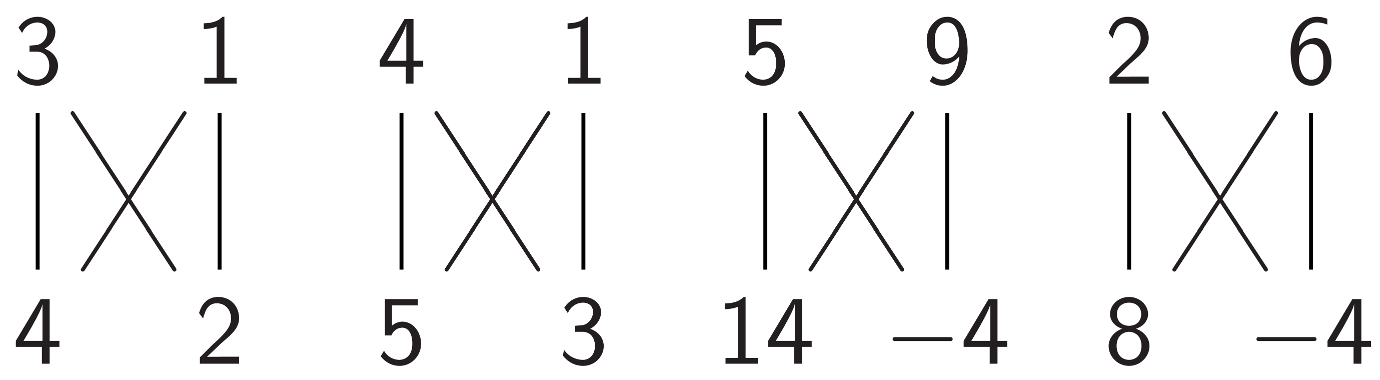
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

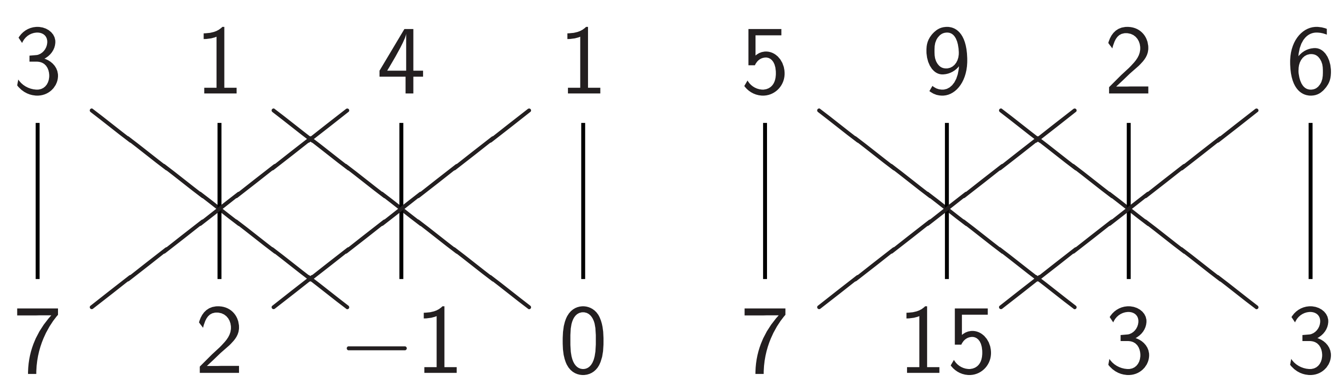
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5g. More shuffling:

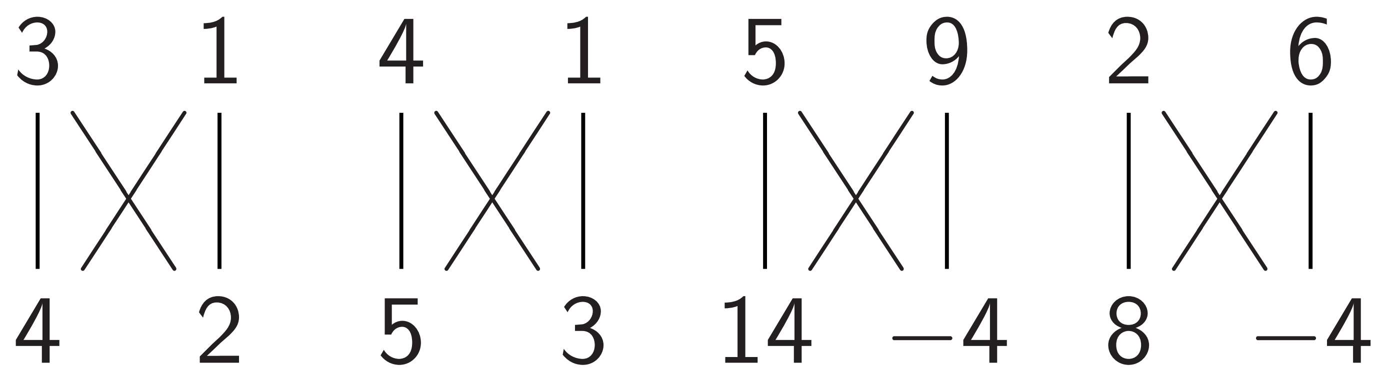
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1.

Each column is a parallel universe performing its own computations.

Hadamard gates

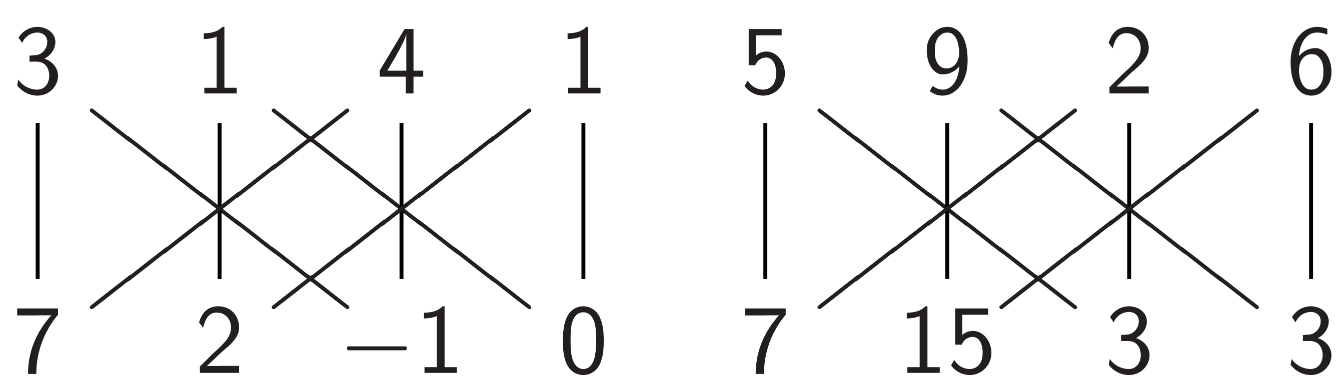
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5h. More shuffling:

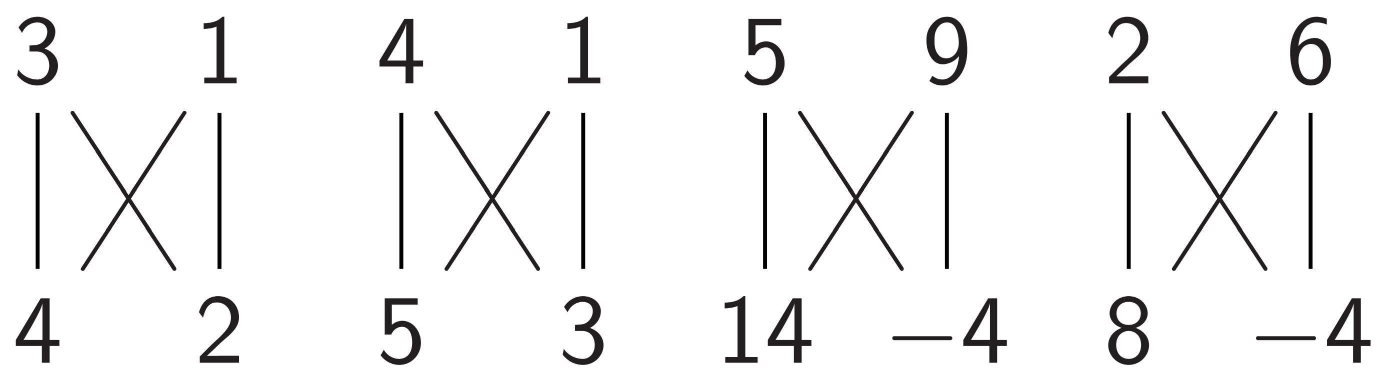
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

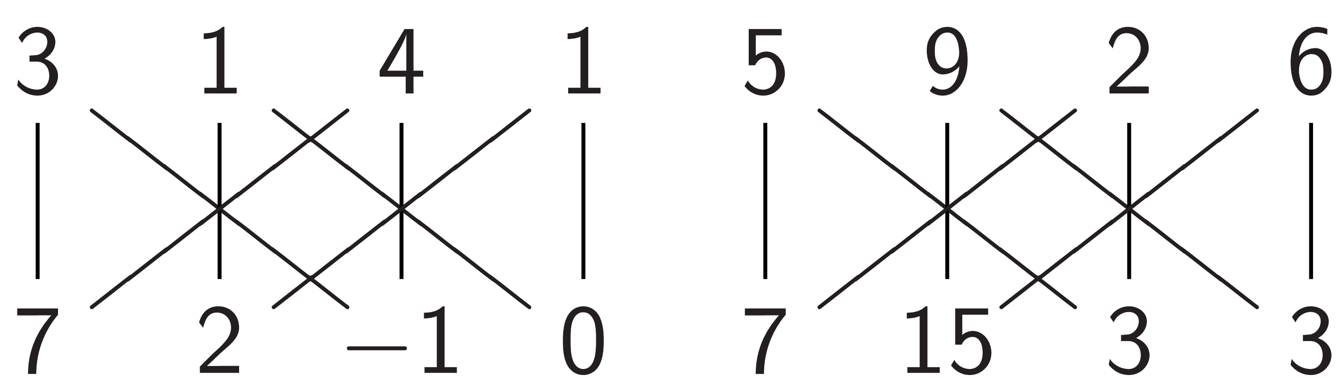
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5i. More shuffling:

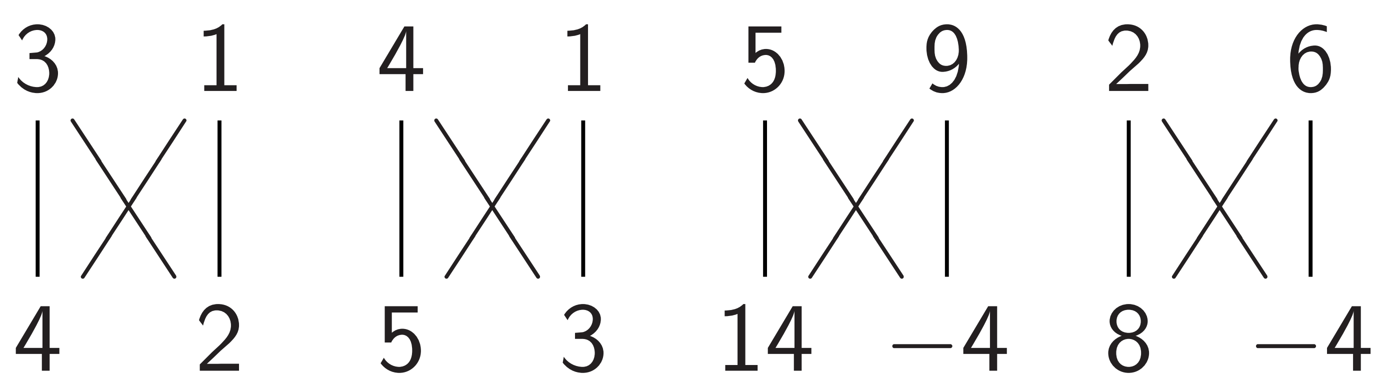
0, 0, 0, 0, 0, 0, 1, 0,
0, 0, 0, 1, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 1,
0, 0, 1, 0, 0, 0, 0, 0,
0, 1, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 1, 0, 0,
1, 0, 0, 0, 0, 0, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

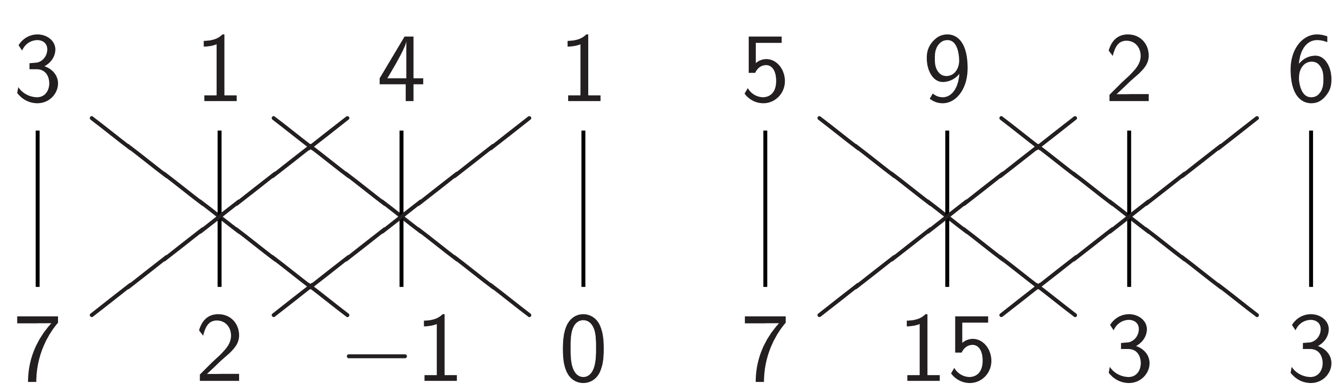
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5j. Final shuffling:

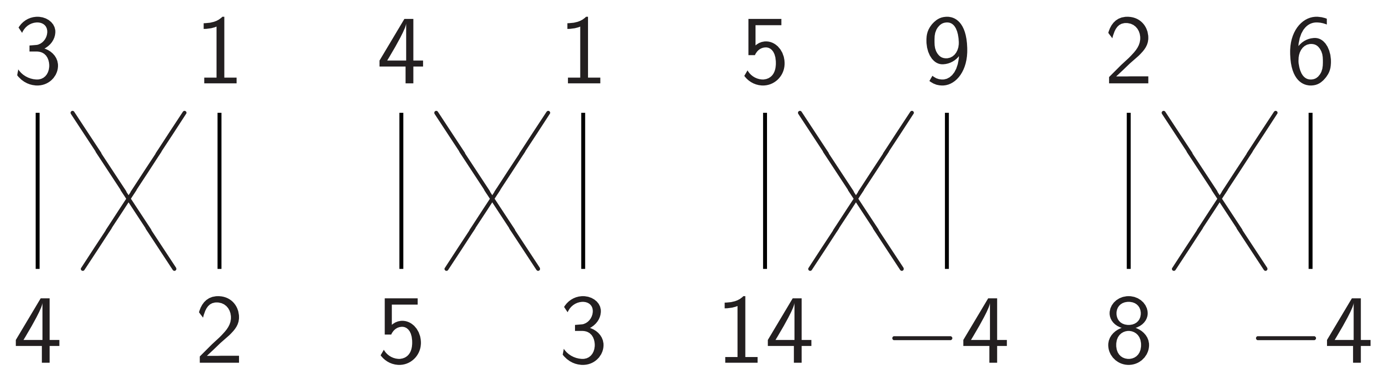
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0.

Each column is a parallel universe performing its own computations.

Hadamard gates

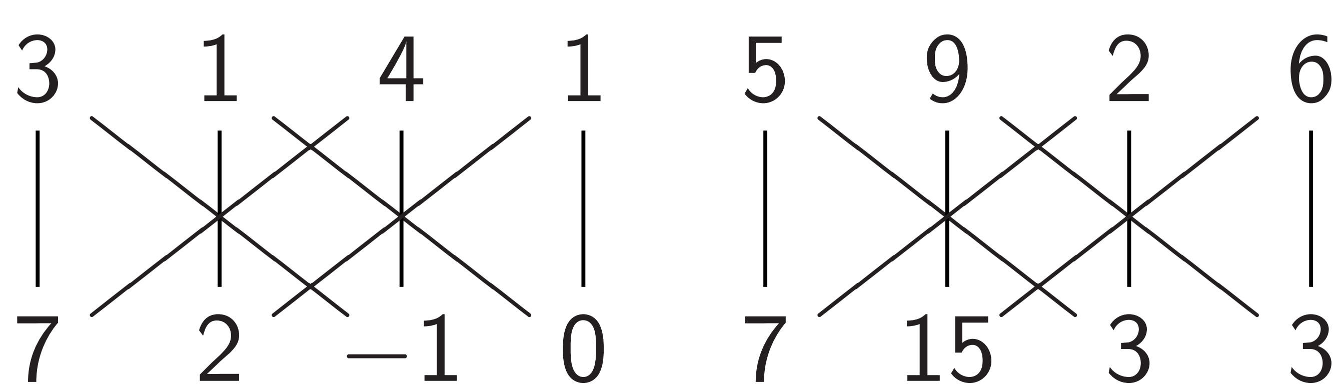
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 5j. Final shuffling:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 1, 0, 0, 1, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 0, 0, 0, 0, 1,
0, 1, 0, 0, 1, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 0, 0, 0, 0, 1, 0, 0.

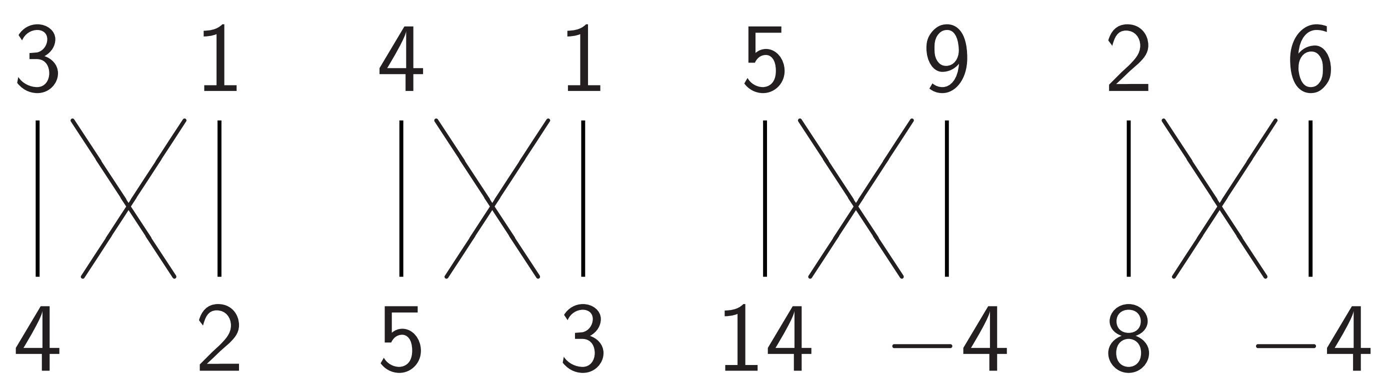
Each column is a parallel universe performing its own computations.

Surprise: u and $u \oplus 101$ match.

Hadamard gates

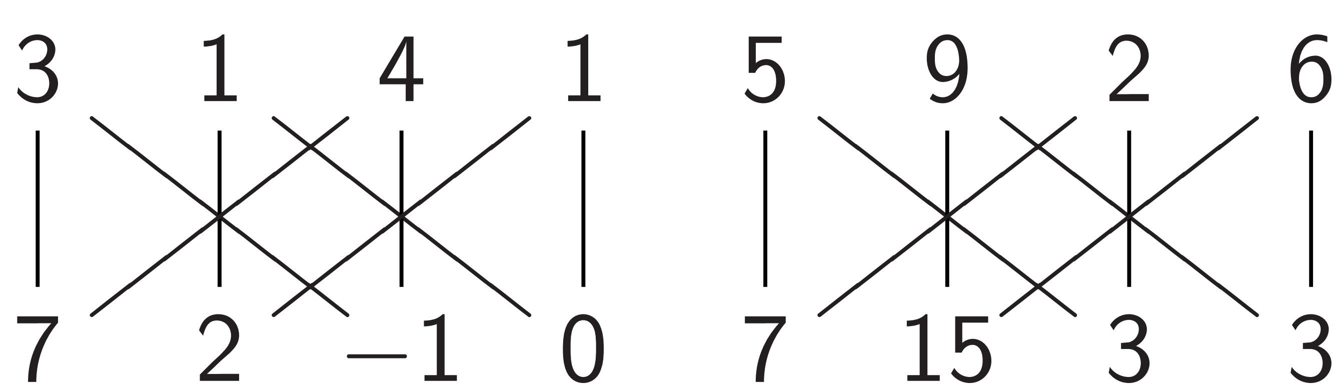
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

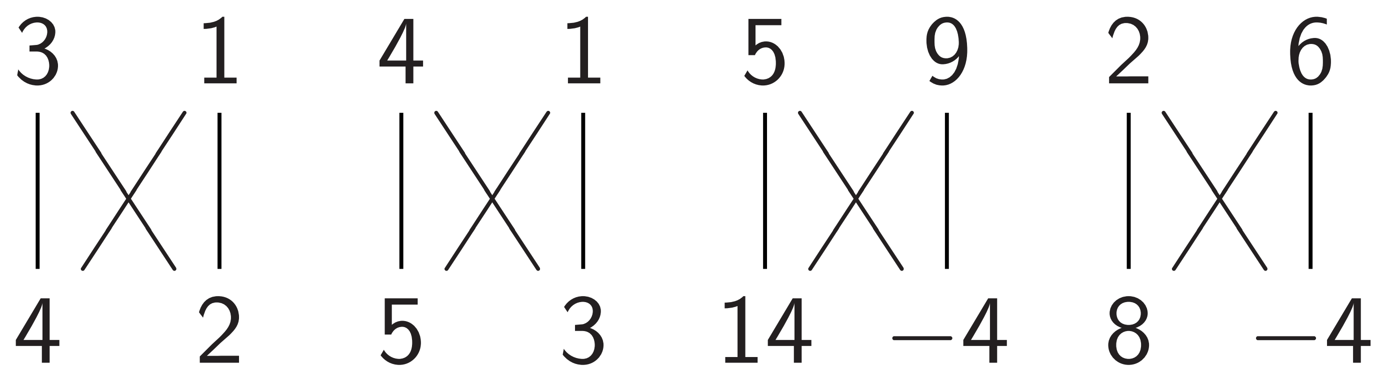
Step 6. Hadamard$_0$:

0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, $\overline{1}$, 0, 0, 1, 1,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 1, 1, 0, 0, 1, $\overline{1}$,
1, $\overline{1}$, 0, 0, 1, 1, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0,
1, 1, 0, 0, 1, $\overline{1}$, 0, 0.

Hadamard gates

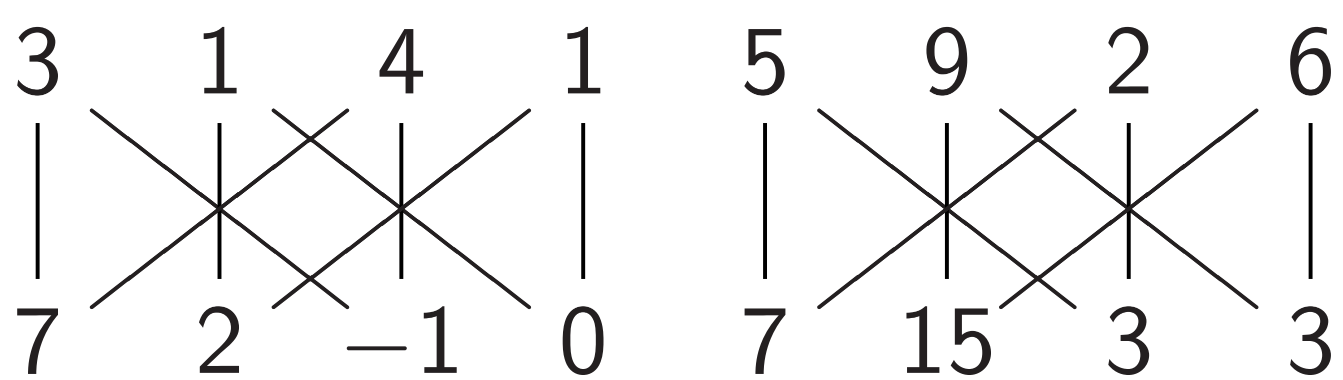
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

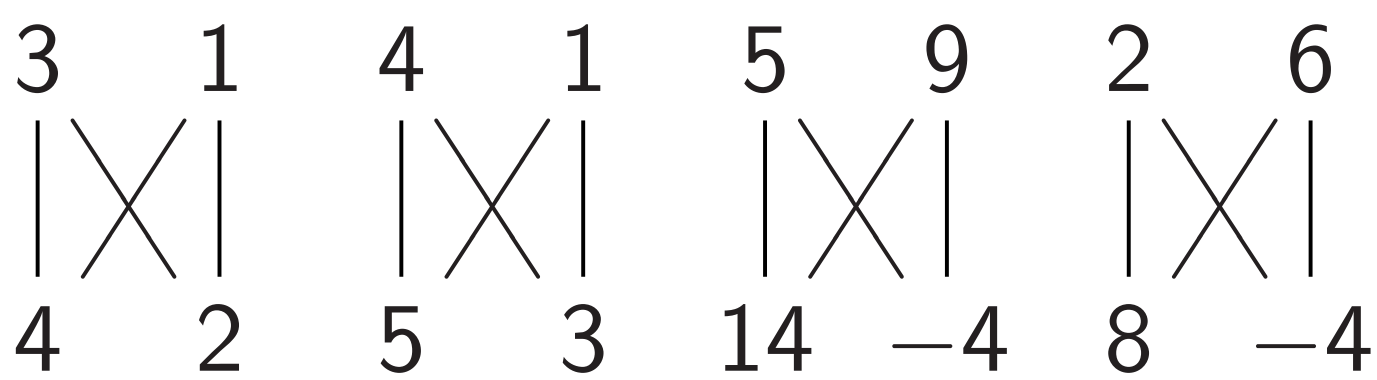
Step 7. Hadamard$_1$:

$0, 0, 0, 0, 0, 0, 0, 0,$
$1, \overline{1}, \overline{1}, 1, 1, 1, \overline{1}, \overline{1},$
$0, 0, 0, 0, 0, 0, 0, 0,$
$1, 1, \overline{1}, \overline{1}, 1, \overline{1}, \overline{1}, 1,$
$1, \overline{1}, 1, \overline{1}, 1, 1, 1, 1,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$1, 1, 1, 1, 1, \overline{1}, 1, \overline{1}.$

Hadamard gates

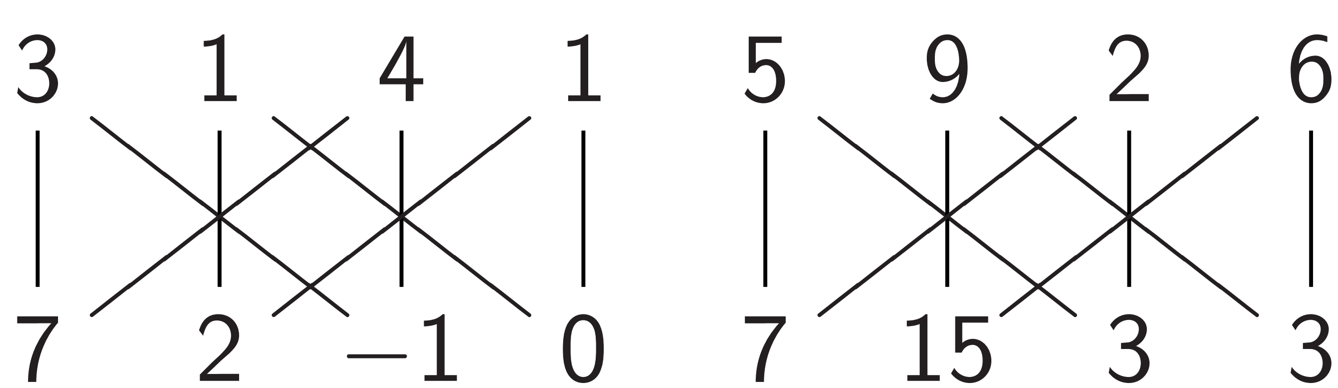
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

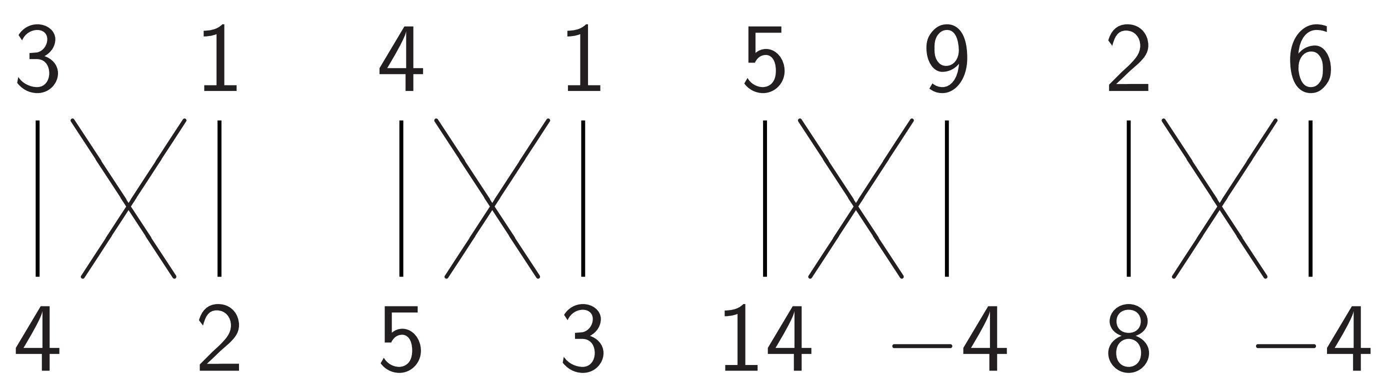
Step 8. Hadamard$_2$:

$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, 2, 0, \overline{2},$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2},$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, 2, 0, 2.$

Hadamard gates

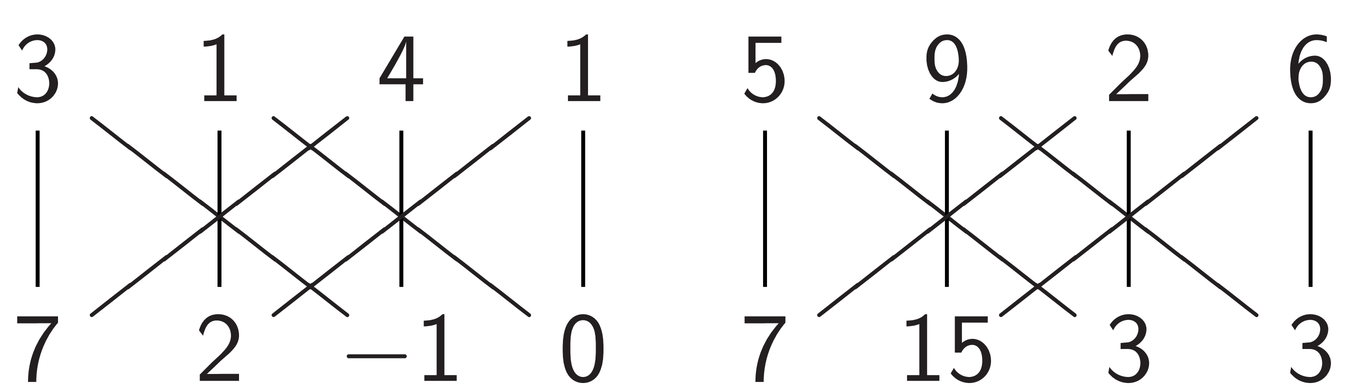
Hadamard$_0$:

$(a, b) \mapsto (a + b, a - b)$.



Hadamard$_1$:

$(a, b, c, d) \mapsto$
$(a + c, b + d, a - c, b - d)$.



Simon's algorithm

Step 8. Hadamard$_2$:

$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, \overline{2}, 0, 2,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, \overline{2}, 0, 0, 2, 0, \overline{2},$
$2, 0, 2, 0, 0, \overline{2}, 0, \overline{2},$
$0, 0, 0, 0, 0, 0, 0, 0,$
$0, 0, 0, 0, 0, 0, 0, 0,$
$2, 0, 2, 0, 0, 2, 0, 2.$

Step 9: Measure. Obtain some information about the surprise: a random vector orthogonal to 101.