IV Quantum Computing and Quantum Algorithms

1. The circuit model

a) Classical computation

Use of classical computers (abstractly):

Solve problems $\equiv$ compute functions

$$f : \{0,1\}^n \longrightarrow \{0,1\}^m$$

$$\underline{x} = (x_1, \dots, x_n) \longmapsto f(x_1, \dots, x_n)$$

The function f depends on the problem we want to solve, $\underline{x}$ encodes the instance of the problem.

E.g.: Problem = multiplication : $(a,b) \longmapsto a \cdot b$

$$\underline{x} = (\underline{x}^1, \underline{x}^2) \longmapsto f(\underline{x}) = \underline{x}^1 \cdot \underline{x}^2$$

encoded in binary

Problem = <u>Factorization</u>:

$\underline{x}$: integer; $f(\underline{x})$: list of prime factors (suitably encoded)

<u>More precisely</u>:

Each problem is encoded by a <u>family</u> of functions $f \equiv f^{(n)} : \{0,1\}^n \longrightarrow \{0,1\}^m$, with $m = \text{poly}(n)$, $n \in \mathbb{N}$ – one for each input size.

i.e.: m grows at most polynomially with n (technically, $\exists \alpha > 0$ s.th. $\frac{m}{n^\alpha} \longrightarrow 0$).

(Technical point: It must be possible to "construct the functions $f^{(n)}$ systematically and efficiently", see later!)

<u>Which ingredients</u> do we need to compute a <u>general function f</u>?

(i) $f : \{0,1\}^n \longrightarrow \{0,1\}^m$

$$f(\underline{x}) = (f_1(\underline{x}), f_2(\underline{x}), \dots, f_m(\underline{x}))$$

where $f_k(\underline{x}) : \{0,1\}^n \longrightarrow \{0,1\}$

$\Rightarrow$ can restrict analysis to boolean functions

$$f : \{0,1\}^n \longrightarrow \{0,1\}.$$

(ii) Define $L = \{\underline{y} \mid f(\underline{y}) = 1\} = \{\underline{y}^1, \underline{y}^2, \dots, \underline{y}^\ell\}$.

Define $\delta_{\underline{y}}(\underline{x}) = \begin{cases} 0 \, ; & \underline{x} \neq \underline{y} \\ 1 \, ; & \underline{x} = \underline{y} \end{cases}$ ← bitwise equality!

Then, $f(\underline{x}) = \delta_{\underline{y}^1}(\underline{x}) \vee \delta_{\underline{y}^2}(\underline{x}) \vee \dots \vee \delta_{\underline{y}^\ell}(\underline{x})$

"$\vee$": logical "<u>or</u>":

($0 \equiv$ "false",
$1 \equiv$ "true")

$0 \vee 0 = 0$
$0 \vee 1 = 1$
$1 \vee 0 = 1$
$1 \vee 1 = 1$

"$\vee$" is <u>associative</u>:

$$a \vee b \vee c := (a \vee b) \vee c = a \vee (b \vee c)$$

and <u>commutative</u>: $a \vee b = b \vee a$.

(iii) Define <u>bitwise</u> δ:

$$\delta_y(x) = \begin{cases} 0 : & y \neq x \\ 1 : & y = x \end{cases}$$

Then,

$$\delta_{\underline{y}}(\underline{x}) = \delta_{y_1}(x_1) \wedge \delta_{y_2}(x_2) \wedge \ldots \wedge \delta_{y_n}(x_n)$$

"$\wedge$": logical "and":

($0 \equiv$ "false",
$1 \equiv$ "true")

$0 \wedge 0 = 0$
$0 \wedge 1 = 0$
$1 \wedge 0 = 0$
$1 \wedge 1 = 1$

"$\wedge$" is <u>associative</u> & <u>commutative</u>;

"$\wedge$" & "$\vee$" are distributive:

$$(a \vee b) \wedge c = (a \wedge c) \vee (b \wedge c).$$

(In essence, same rules as $\wedge \rightarrow \cdot$, $\vee \rightarrow +$)

(iv)

$$\delta_y(x) = \begin{cases} x & \text{if } y = 1 \\ \neg x & \text{if } y = 0 \end{cases}$$

Logical "not": $\neg 0 = 1$

$\neg 1 = 0$

Combine (i) – (iv):

Any $f(\underline{x})$ can be constructed from 4 ingredients:

"and", "or", "not" gates,

plus a "copy" gate $x \longmapsto (x, x)$.

This is called a universal gate set.

(Note: In fact, already either $\neg(x \wedge y)$ "nand",

or $\neg(x \vee y)$ "nor" are universal, together

with "copy".)

This gives rise to the

Circuit model of computation:

The functions $f \equiv f^{(c)}$ which we can compute

are constructed by concatenating gates from a simple universal gate set (e.g. and/or/not/copy) sequentially in time (i.e., there are no loops allowed). This gives rise to a circuit for $f^{(n)}$.

The difficulty ("computational hardness") of a problem in the circuit model is measured by the number $K(n)$ of elementary gates needed to compute $f^{(n)}$ ($\hat{=}$ # of time steps).

We often distinguish two qualitatively different regimes:

$K(n) \sim \text{poly}(n)$: efficiently solvable (class P)
easy problem

$K(n) \gg \text{poly}(n)$ — e.g. $K(n) \sim \exp(n^\alpha)$:
hard problem

(Technical note: We must suppose that the circuits used for $f^{(n)}$ are <u>uniform</u>, i.e. they can be generated efficiently – e.g. by a simple n-independent computer program. More formally, $f^{(n)}$ should be generated by a Turing machine.)

<u>Example:</u>

f = <u>Multiplication</u>:

Efficient:

$$\begin{array}{r} \overbrace{10110}^{\ell} \times \overbrace{10011}^{\ell'} \\ \hline 10110 \\ 10110 \\ 10110 \\ \hline \underbrace{110100010}_{\ell \times \ell' \text{ additions:}} \end{array} \left.\vphantom{\begin{array}{r}1\\1\\1\end{array}}\right\} \ell'$$

$\ell \times \ell'$ additions:

$O(\ell\ell') \sim O(n^2)$ gates.

f: <u>Factorization</u>.

E.g.: Sieve of Eratosthenes:

$\{0,1\}^n \longrightarrow$ try about $\sqrt{2^n} \sim 2^{n/2}$ cases

$\Longrightarrow$ hard / exp. scaling.

No efficient algorithm known!

Is a typical problem easy or hard?

$$f : \{0,1\}^n \longrightarrow \{0,1\}$$

# of different f: $2^{\underbrace{(2^n)}_{\text{\# of inputs}}}$

$f(\underline{x}) = \begin{cases} 0 \\ 1 \end{cases}$ for each input

But: there are only $c^{\text{poly}(n)}$ circuits of length poly(n)!

# of elem. gates

$\Rightarrow$ As n gets large, most f cannot be computed efficiently (i.e. with poly(n) operations).

Does the computational power depend on the gate set?

NO! By definition, any universal gate set can simulate any other gate set with constant overhead!

<u>Remark</u>: There is a wide range of alternative models of computation, some more and some less realistic:

- CPU
- parallel computers
- "Turing machines" – tape + read/write head
- cellular automata
- ... and lots of exotic models ...

<u>But</u>: All known "reasonable" models of computation can simulate each other with poly(n) overhead

$\Longrightarrow$ same computational power (in the sense above).

<u>Church-Turing-Thesis</u>: All reasonable models of computation have the same computational power.

b) Reversible circuits

For quantum computing – coming soon – we will use the circuit model.

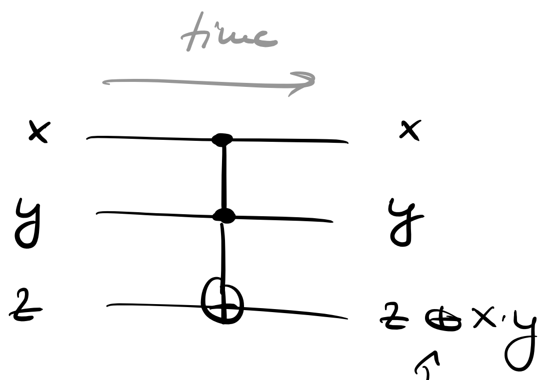
Gates will be replaced by unitaries.

But: Unitaries are reversible, while classical gates (and, or) are irreversible.

Could such a model even do classical computations – – i.e., can we find a universal gate set with only reversible gates?

YES! – Classical computation can be made reversible:

Toffoli gate:



"XOR" = addition mod 2:

$0 \oplus 0 = 0$

$0 \oplus 1 = 1$

$1 \oplus 0 = 1$

$1 \oplus 1 = 0$

also assoc., commu., & distr. w/ $\wedge$. (like "∨")

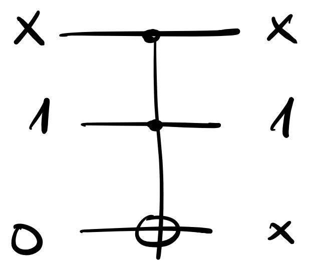
→ Toffoli gate is reversible

(it is its own inverse, since $(z \oplus x \cdot y) \oplus x \cdot y = z$)

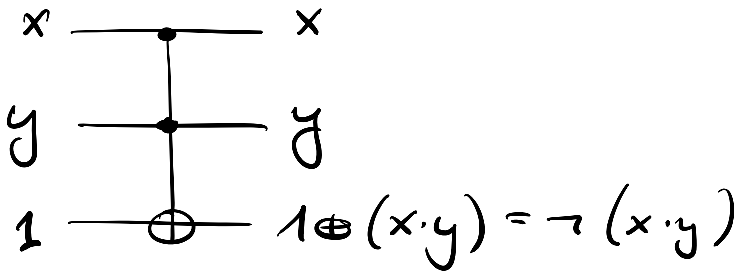
→ Toffoli gate can simulate and/or/not/copy,

by using ancillas in state "0" or "1":

E.g.:



"copy"



"Nand"

⟹ gives reversible universal gate set

(but requires ancillas)

This can be used to compute any $f(\underline{x})$ reversibly,

using ancillas, with essentially the same # of gates:

$$f^R(\underline{x}, y) \longmapsto (\underline{x}, f(\underline{x}) \oplus y)$$

bitwise XOR.

(Idea: Replace any gate by a reversible gate using ancillas. Then XOR the result into the y register. Finally, run the circuit backwards to "uncompute" the ancillas. Ancilla count can be optimized for → cf. Preskill's notes.)

⇒ Everything can be computed reversibly.

BUT: 3-bit gate is required!

(→ Homework)

c) Quantum Circuits

Most common model for quantum computation:

The circuit model:

- Quantum system consisting of qudits: tensor product structure.
- Universal gate set $S = \{U_1, \dots, U_k\}$ of few-qudit gates (typ. 1- and 2-qudit gates) U_j. (See later for definition of "universal"!)
- Construct circuits by sequentially applying

elements of S to a subset of qubits:

$$|\psi_{out}\rangle = U_T U_{T-1} \cdot \ldots \cdot U_1 |\psi_{in}\rangle$$

U_j acting on subset of qubits

- Initial state:

$$|\psi_{in}\rangle = |x_1\rangle|x_2\rangle \ldots |x_n\rangle \overbrace{|0\rangle|0\rangle \ldots |0\rangle}^{\ell}$$

$$= |\underline{x}\rangle |\underline{0}\rangle$$

ancillas

encodes instance of problem

  - alternatively, we can also have

$$|\psi_{in}\rangle = |\underline{0}\rangle \equiv |0\rangle^{\otimes \ell}$$

and encode the instance in the circuit.

- At the end of the computation, measure the final state $|\psi_{out}\rangle$ in the computational basis $\{|0\rangle, |1\rangle\}$

$\longrightarrow$ outcome $|\underline{y}\rangle$ w/ prob. $p(\underline{y}) = |\langle \underline{y}|\psi_{out}\rangle|^2$

<u>Notes</u>: • This is a <u>probabilistic</u> scheme – it outputs y w/ some prob. $p(y)$. In principle, we should compare to <u>class. probabilistic</u> schemes – see later.

• We need not measure all qubits –

not measuring = tracing = measuring and ignoring outcome

• POVMs don't help – we can simulate them (→ Naimark). Similarly, CP maps don't help – we can simulate them (Stinespring + trace ancilla).

• Measurements at earlier times don't help: Can always postpone them (they commute). If gate at later time would depend on meas. outcome: This dependence can be realized <u>inside</u> the circuit w/ "controlled gates"

(cf. later + homework)

What gate set should we choose?

- There is a continuum of gates – situation much more rich.

- Different notions of universality exist:

  – exact universality: Any n-qubit gate can be realised exactly.

    → Requires a continuous family of universal gates (counting argument!)

  – approximate universality: Any n-qubit gate can be approximated well by gate set

    (Finite gate set sufficient;

    Solovay-Kitaev-Theorem: ε-approximation (in $\|\cdot\|_\infty$-Norm) of 1-qubit gate requires

    $O(\mathrm{poly}(\log(1/\varepsilon)))$ gates from a suitable finite set.)

- 1- and -2 qubit gates alone are universal!

  (cf. classical: 3-bit gates needed!!)

- For approximate universality, almost any ~~single~~ two-qubit gate will do!

  w/ prob. 1.

- More univ. sets: later!

d) Universal gate set

Our exact universal gate set:

(i) 1-qubit rotations about X & Z axis:

$$R_X(\phi) = e^{-iX\phi/2} \quad ; \quad X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix}, \quad X^2 = I$$

$$R_Z(\phi) = e^{-iZ\phi/2} \quad ; \quad Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}, \quad Z^2 = I.$$

For $M^2 = I$: $\quad e^{-iM\phi/2} = \cos\phi/2 \; I - i\sin\phi/2 \; M$

$$\Longrightarrow \quad R_X(\phi) = \begin{pmatrix} \cos\phi/2 & -i\sin\phi/2 \\ -i\sin\phi/2 & \cos\phi/2 \end{pmatrix}$$

$$R_Z(\phi) = \begin{pmatrix} e^{-i\phi/2} & 0 \\ 0 & e^{i\phi/2} \end{pmatrix}$$

Can be understood as rotations on Bloch sphere about X/Z axis by angle ϕ (i.e., rotations in $SO(3) \cong SU(2)/\mathbb{Z}_2$).

Together, R_x and R_z generate all rotations in $SO(3)$ (Euler angles!), and thus in $SU(2)$ up to a phase.

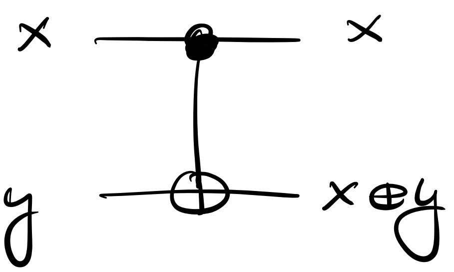
Lemma: For any $U \in SU(2)$,

$$U = e^{i\phi} R_x(\alpha) R_z(\beta) R_x(\gamma) \quad \text{for some } \phi, \alpha, \beta, \gamma.$$

Proof: Homework.

(ii) one two qubit gate (almost all would do!).

Typically, we use "controlled-NOT" = "CNOT":

CNOT = 

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}$$

CNOT flips y iff $x = 1$: classical gate!

<u>Can prove:</u> This gate set can create any n-qubit U <u>exactly</u> (but of course not efficiently – U has $\sim (2^n)^2 = 4^n$ real parameters).

Overview of a number of important gates & identities

(Proof/check: Homework!)

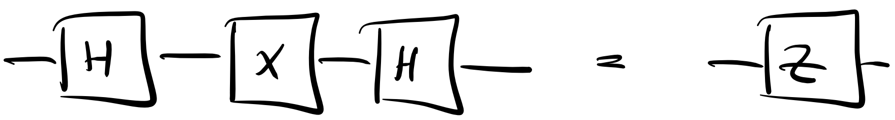
Hadamard gate: $$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

$$H = H^\dagger ; \quad H^2 = I .$$

$$H R_x(\phi) H = R_z(\phi)$$

$$H R_z(\phi) H = R_x(\phi)$$

<u>Graphical "circuit" notation:</u>

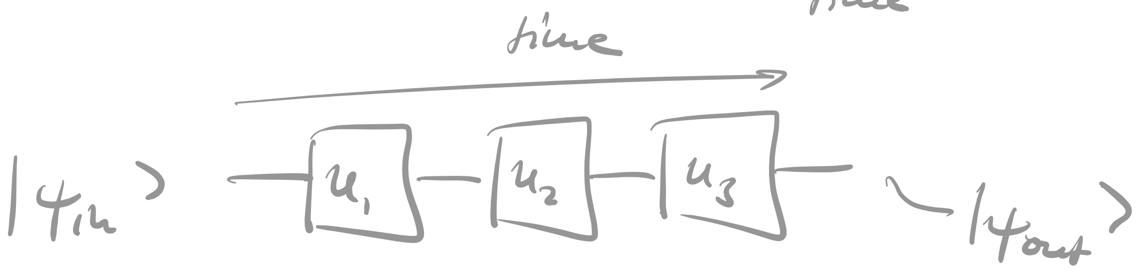


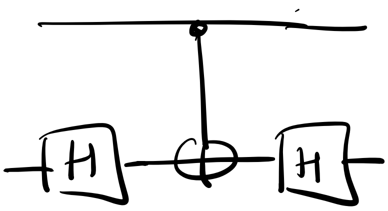
<u>Important:</u>

Matrix notation: time goes right to left

Circuit notation: time goes left to right:

i.e.: $|\psi_{in}\rangle \longmapsto |\psi_{out}\rangle = \underbrace{U_3 U_2 U_1}_{\xleftarrow{\text{time}}} |\psi_{in}\rangle$

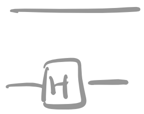




$=$

$= \underbrace{\begin{pmatrix} 1 & & & \\ & 1 & & \\ & & 1 & \\ & & & -1 \end{pmatrix}}$

only applied to 2nd qubit, i.e.:

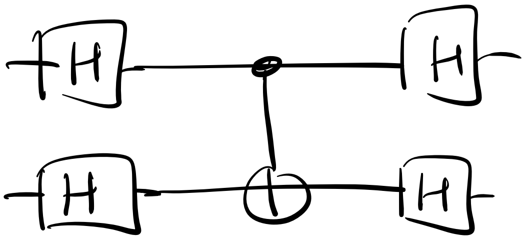


$\equiv I \otimes H.$

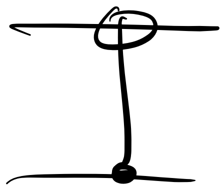
"Controlled-Z"

"Controlled-Phase"

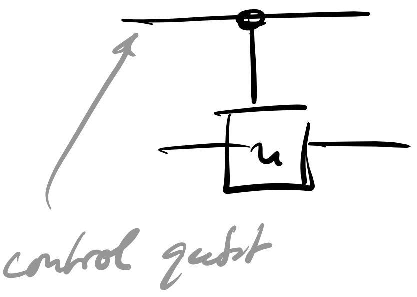
CZ, CPHASE



$=$



Generally: For a unitary $U \in SU(2)$,

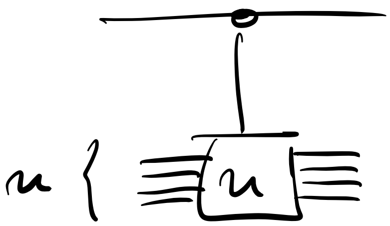


$$= \begin{pmatrix} I & 0 \\ 0 & U \end{pmatrix} \quad \text{"controlled-}U\text{"}$$

control $= |0\rangle$ control $= |1\rangle$

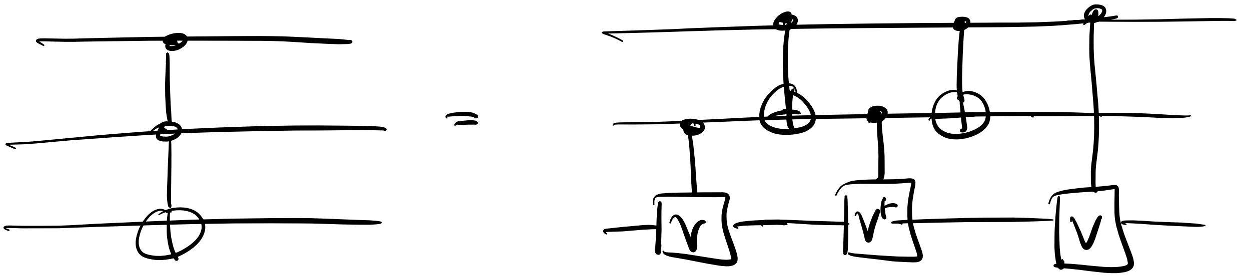
Can be implemented w/ 2 CNOT (→ HW!)

Also for $U \in SU(2^n)$:



$$= \begin{pmatrix} I_{2^n} & 0 \\ 0 & U \end{pmatrix}$$

Circuit for Toffoli:

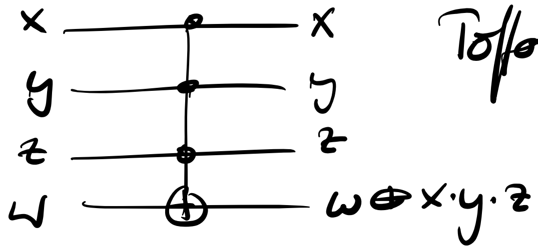


with $V = \frac{1-i}{2}(I + iX)$

U to controlled-U:

Given circuit for U – in particular, a classical reversible circuit – we can also build controlled-U:

Just replace every gate by its controlled-version, in particular Toffoli by



Toffoli w/ 3 controls can be built from normal Toffoli (since class. universal!)

Finally, some <u>further approx. universal gate sets:</u>

- CNOT + 2 random 1-qubit gates
- CNOT + H + $T = R_z(\pi/4)$ ("$\pi/8$ gate")

2. Oracle-based algorithms

a) The Deutsch algorithm

Consider $f: \{0,1\} \longrightarrow \{0,1\}$

Let f be "very hard to compute" – e.g. long circuit

Want to know: Is $f(0) = f(1)$?

(e.g.: will a specific chess move affect result?)

How often do we have to run the circuit for f

(= "evaluate f")? — We think of f as a "black box"

or "oracle": How many oracle queries are needed?

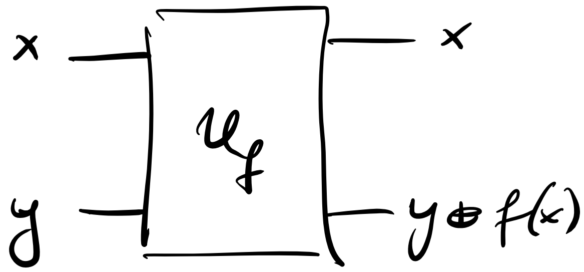
Classically, we clearly need 2 queries:

Compute $f(0)$ and $f(1)$.

Can quantum physics help?

Consider reversible implementation of f:

$$f^R : (x, y) \longmapsto (x, y \oplus f(x))$$

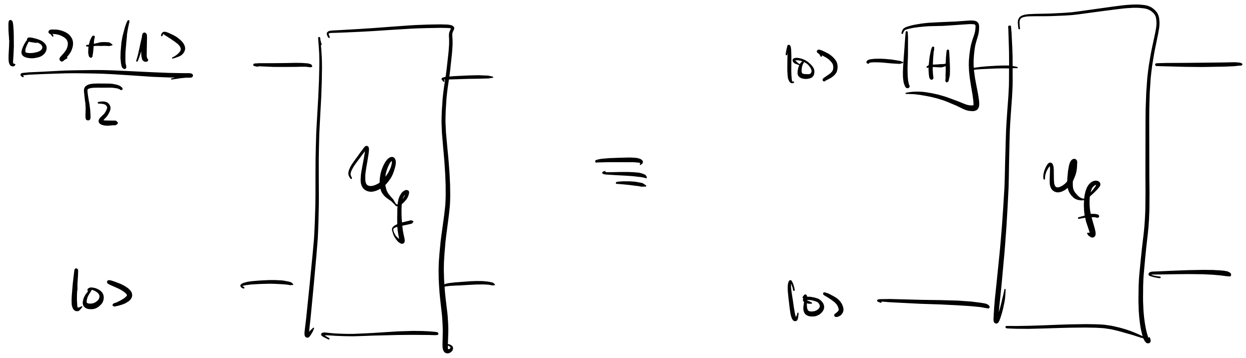


$$|x\rangle|y\rangle \longmapsto |x\rangle|y \oplus f(x)\rangle$$

Of course, we can use U_f to compute $f(0)$ or $f(1)$ on a quantum computer, but this we could also do classically. So, can we do better than this?

Try to use <u>superpositions as inputs</u>?

<u>First attempt:</u>



$$\frac{|0\rangle+|1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{1}{\sqrt{2}}\Big(|0\rangle|0\rangle + |1\rangle|0\rangle\Big) \xmapsto{U_f} \frac{1}{\sqrt{2}}\Big(|0\rangle|f(0)\rangle + |1\rangle|f(1)\rangle\Big)$$

$\longrightarrow$ Have evaluated f on <u>both outputs</u>!

But how can we <u>extract the relevant information</u> from this state (i.e. do a <u>measurement</u>)?

- Meas. in comp. basis: <u>collapse</u> superpos. to <u>one case</u>!
- More generally: If <u>$f(0) \neq f(1)$</u>, the output is in

$$S_{\neq} = \left\{ \tfrac{1}{\sqrt{2}}\left(|0\rangle|0\rangle + |1\rangle|1\rangle\right), \tfrac{1}{\sqrt{2}}\left(|0\rangle|1\rangle + |1\rangle|0\rangle\right) \right\},$$

and for <u>$f(0) = f(1)$</u> in

$$S_{=} = \left\{ |+\rangle|0\rangle, |+\rangle|1\rangle \right\}.$$

$\Longrightarrow$ not orthogonal, i.e. not (determ.) distinguishable!

But: We can do measurements which, with some probability, allow to conclude that $f(0) = f(1)$ or $f(0) \neq f(1)$.

E.g., all states in $S_=$ are orth. to

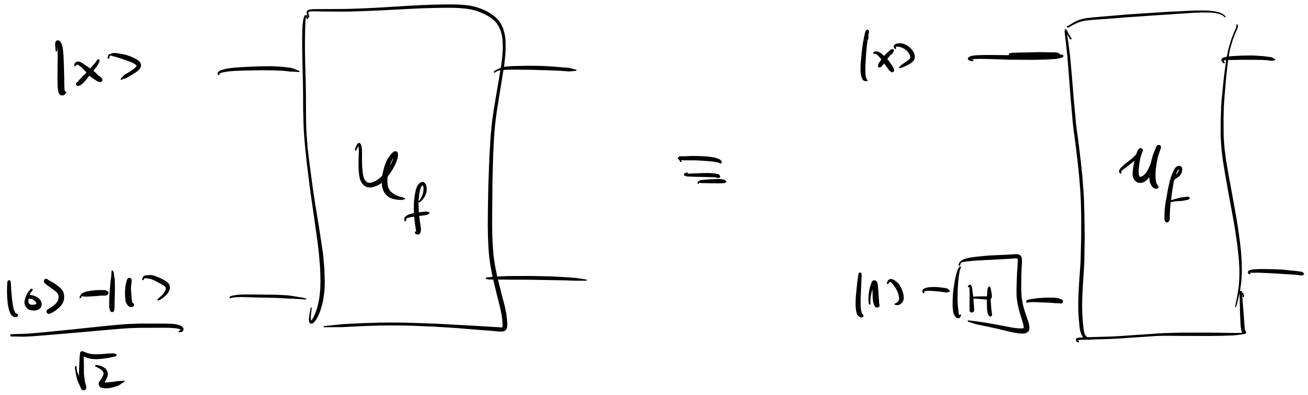
$R_{\neq} = \{|-\rangle|0\rangle, |-\rangle|1\rangle\}$, and all states in $S_{\neq}$ to $R_= = \left\{\tfrac{1}{\sqrt{2}}\left(|0\rangle|0\rangle - |1\rangle|1\rangle\right), \tfrac{1}{\sqrt{2}}\left(|0\rangle|1\rangle - |1\rangle|0\rangle\right)\right\}$.

$\Longrightarrow$ A <u>POVM</u> which includes those outcomes plus an extra "fail" outcome allows to unambiguously identify whether $f(0) \stackrel{?}{=} f(1)$, with some probability

Optimal success probability: $\frac{1}{2}$ ($\rightarrow$ Homework)

While that is impossible classically, it does not give an improvement on average.

Second attempt:



$$|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \xmapsto{U_f} |x\rangle\left(\frac{|f(x)\rangle - |1\oplus f(x)\rangle}{\sqrt{2}}\right) =$$

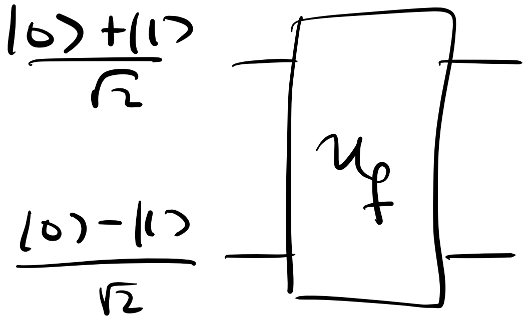
$$= \begin{cases} f(x)=0: & |x\rangle \dfrac{|0\rangle - |1\rangle}{\sqrt{2}} \\ f(x)=1: & |x\rangle \dfrac{|1\rangle - |0\rangle}{\sqrt{2}} \end{cases}$$

$$= |x\rangle\left[(-1)^{f(x)} \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$$

$$= (-1)^{f(x)} |x\rangle \left(\frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

Not useful by itself: $f(x)$ only encoded in <u>global phase</u> for each <u>classical input</u> <u>$|x\rangle$</u>.

<u>Combine attempts:</u>



:

$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} = \frac{1}{\sqrt{2}} \left(|0\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} + |1\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

$$= \frac{1}{\sqrt{2}} \left((-1)^{f(0)} |0\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} + (-1)^{f(1)} |1\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} \right)$$

$$= \frac{(-1)^{f(0)} |0\rangle + (-1)^{f(1)} |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

<u>Observations:</u>

→ No entanglement created (!)

→ 2nd qubit – the one where U_f outputs the function value – is unchanged (!!)

→ 1st qubit gets a phase $(-1)^{f(x)}$

("phase kick-back technique")

State of 1st qubit:

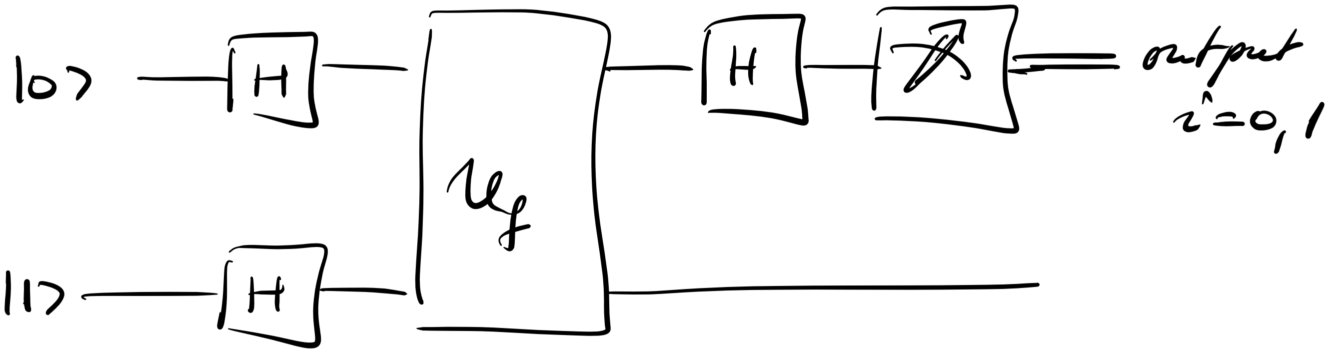
$$f(0) = f(1) \quad \Longleftrightarrow \quad \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$f(0) \neq f(1) \quad \Longleftrightarrow \quad \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$

(up to irrelevant global phase)

Orthogonal states! ⟹ measurement of 1st qubit in basis $\{|+\rangle, |-\rangle\}$ (or apply –[H]– & measure in $\{|0\rangle, |1\rangle\}$) allows to decide if $f(0) \stackrel{?}{=} f(1)$!

Deutsch algorithm:



output $i=0$: $\Rightarrow$ $f(0) = f(1)$

$i=1$: $\Rightarrow$ $f(0) \neq f(1)$

<u>One application</u> of U_f has been sufficient!

$\Rightarrow$ Speed-up compared to <u>class. algorithm</u>

(1 vs. 2 oracle queries).

<u>Interesting to note:</u> 2nd qubit never needs to be measured – and it contains <u>no information</u>.

<u>Two main insights:</u>

- Use input $\sum |x\rangle$ to <u>evaluate f on all inputs simultaneously</u>.

- This parallelism alone is not enough – <u>need</u> a <u>smart</u> way to <u>read out the relevant information</u>.

However, a constant speed-up is not that impressive – in particular, it is highly architecture-dependent!

<u>Thus:</u>

b) The Deutsch-Jozsa algorithm

Consider $f:\{0,1\}^n \longrightarrow \{0,1\}$ with promise (i.e., a condition we know is met by f) that

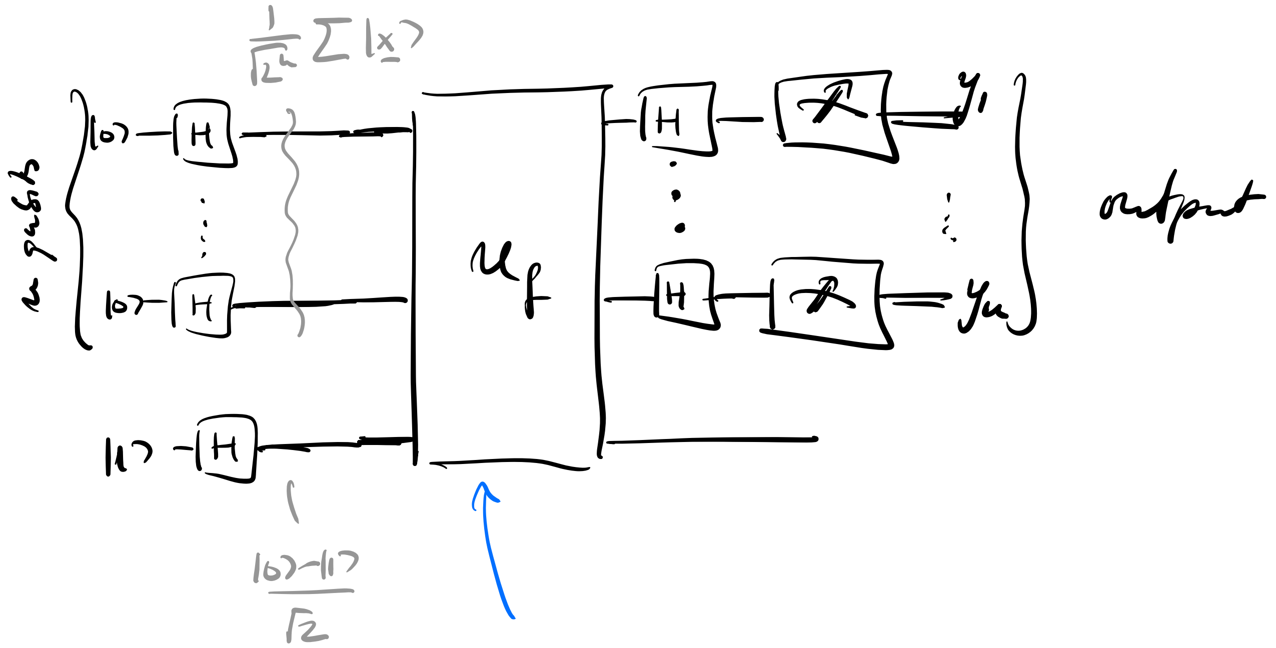
either $f(\underline{x}) = c \quad \forall \underline{x}$ ("f constant")

or $\left|\{\underline{x} \,|\, f(\underline{x})=0\}\right| = \left|\{\underline{x} \,|\, f(\underline{x})=1\}\right|$ ("f balanced")

Want to know: Is f constant or balanced?

How many queries needed?

Use same idea: Input $\sum |x\rangle$ and $\frac{|0\rangle - |1\rangle}{\sqrt{2}}$:



$U_f: |\underline{x}\rangle|y\rangle \longmapsto |\underline{x}\rangle|y \oplus f(x)\rangle$

Before analyzing circuit: What is action of $H^{\otimes 4}$?

$$H: \ |x\rangle \longmapsto \frac{1}{\sqrt{2}} \sum_{y=0,1} (-1)^{x \cdot y} |y\rangle$$

$$H^{\otimes n}: \ |x_1, \dots, x_n\rangle \longmapsto \frac{1}{\sqrt{2^n}} \sum_{\underline{y}} (-1)^{x_1 y_1} \cdot \ldots \cdot (-1)^{x_n y_n} |y_1, \dots, y_n\rangle$$

or:
$$|\underline{x}\rangle \longmapsto \frac{1}{\sqrt{2^n}} \sum (-1)^{\underline{x} \cdot \underline{y}} |\underline{y}\rangle$$

where $\underline{x} \cdot \underline{y} := x_1 y_1 \oplus x_2 y_2 \oplus \dots \oplus x_n y_n$

("scalar product" mod 2).

is NOT a scalar product!

Analysis of circuit:

we omit normalization!

$$|\underline{0}\rangle|1\rangle \xrightarrow{H^{\otimes n} \otimes H} \Big(\sum_{\underline{x}} |\underline{x}\rangle\Big)\big(|0\rangle - |1\rangle\big)$$

phase kick-back

$$\xrightarrow{U_f} \Big(\sum_{\underline{x}} (-1)^{f(\underline{x})} |\underline{x}\rangle\Big)\big(|0\rangle - |1\rangle\big)$$

$$\xrightarrow{H^{\otimes n} \otimes I} \Big(\sum_{\underline{y}} \underbrace{\sum_{\underline{x}} (-1)^{f(\underline{x}) + \underline{x} \cdot \underline{y}}}_{=: a_{\underline{y}}} |\underline{y}\rangle\Big)\big(|0\rangle - |1\rangle\big)$$

$p_y := |a_y|^2$ is the probability to measure $\underline{y} = (y_1, \dots, y_n)$.

<u>f constant:</u> $f(\underline{x}) = c$

$$a_y = (-1)^c \underbrace{\sum_{\underline{x}} (-1)^{\underline{x} \cdot \underline{y}}}_{\propto\, \delta_{\underline{y},\underline{0}}} = (-1)^c \delta_{\underline{y},\underline{0}}$$

<u>f balanced:</u>

For $\underline{y} = \underline{0}$: $a_{\underline{0}} = \sum_{\underline{x}} (-1)^{f(\underline{x}) + \underline{x} \cdot \underline{0}}$

$$= \sum_{x} (-1)^{f(\underline{x})} = 0$$

<u>f balanced!</u>

<u>Thus:</u>

Output $\underline{y} = \underline{0}$ $\Longrightarrow$ f constant

Output $\underline{y} \neq \underline{0}$ $\Longrightarrow$ f balanced

$\Longrightarrow$ We can <u>unambiguously</u> <u>distinguish</u> the 2 cases with <u>one query to the oracle for f!</u>

What is the speed-up vs. classical methods?

Quantum: 1 use of f.

Classical: Worst case, we have to determine

$2^{n-1}+1$ values of f to be sure!

$\Rightarrow$ exponential vs. constant!

But: If we are ok to get right answer with very high probability $p = 1 - p_{error}$, then for k queries to f,

$$p_{error} \approx 2 \cdot \underbrace{\left(\frac{1}{2}\right)^k}$$

$\approx$ prob. to get $k\times$ same outcome for balanced f, if $k \ll 2^n$.

i.e.: $k \sim \log(1/p_{error})$.

Randomized classical: Much smaller speed-up vs. randomized classical algorithm (even for exp. small error, $k \sim n$ oracle calls are sufficient.)

c) Simon's algorithm

... will give us a true exponential speedup
(also rel. to <u>randomized</u> class. algorithms)
in terms of oracle queries!

<u>Oracle</u>: $f: \{0,1\}^n \to \{0,1\}^n$

with <u>promise</u>:

$\exists \underline{a} \neq \underline{0}$ s.th. $f(\underline{x}) = f(\underline{y})$ <u>exactly if</u> $\underline{y} = \underline{x} \oplus \underline{a}$.

("hidden periodicity")

<u>Task</u>: Find $\underline{a}$ by querying f.

<u>Classical</u>: Need to query $f(\underline{x}_i)$ until pair $\underline{x}_i, \underline{x}_j$ with $f(\underline{x}_i) = f(\underline{x}_j)$ is found.

Roughly: k queries $x_1, \ldots, x_k \longrightarrow \sim k^2$ pairs,

for each pair: $\text{prob}\left(f(\underline{x}_i) = f(\underline{x}_j)\right) \approx 2^{-n}$

$\Longrightarrow P_{\text{success}} \sim k^2 \, 2^{-n}$

$\Rightarrow$ need $k \sim 2^{cn}$ queries!

Quantum algorithm (Simon's algorithm):

i) Start with $\frac{1}{\sqrt{2^n}} \sum_{\underline{x}} |\underline{x}\rangle = H^{\otimes n} |\underline{0}\rangle$

ii) Apply U_f : $|\underline{x}\rangle|y\rangle \longmapsto |\underline{x}\rangle|y \oplus f(\underline{x})\rangle$

$$U_f : \left(\frac{1}{\sqrt{2^n}} \sum_{\underline{x}} |\underline{x}\rangle_A \right) |\underline{0}\rangle_B \longmapsto \frac{1}{\sqrt{2^n}} \sum_{\underline{x}} |\underline{x}\rangle_A |f(\underline{x})\rangle_B$$

iii) Measure B. $\Longrightarrow$ Collapse onto random $f(\underline{x}_0)$ (and thus random $\underline{x}_0$).

$\Longrightarrow$ Register A collapses onto

$$\frac{1}{N} \sum_{\underline{x} : f(\underline{x}) = f(\underline{x}_0)} |x\rangle = \frac{1}{\sqrt{2}} \left(|\underline{x}_0\rangle + |\underline{x}_0 \oplus \underline{a}\rangle \right)$$

— How can we extract a? —

(Meas. in comp. basis $\to$ collapse on rand. x_0 : useless.)

iv) Apply $H^{\otimes n}$ again:

$$H^{\otimes n} |\underline{x}\rangle \propto \sum_{\underline{y}} (-1)^{\underline{x} \cdot \underline{y}} |\underline{y}\rangle$$

$$H^{\otimes n} \left(\frac{1}{\sqrt{2}} \left(|\underline{x}_0\rangle + |\underline{x}_0 \oplus a\rangle \right) \right)$$

$$= \frac{1}{\sqrt{2^{n+1}}} \sum_{y} \underbrace{\left[(-1)^{\underline{x}_0 \cdot y} + (-1)^{(\underline{x}_0 + \underline{a}) \cdot y} \right]}_{\substack{\underline{a}\cdot y = 0 \;\Rightarrow\; = 2\cdot(-1)^{\underline{x}_0 \cdot y} \\ \underline{a}\cdot y = 1 \;\Rightarrow\; = 0}} |y\rangle$$

$$= \frac{1}{\sqrt{2^{n-1}}} \sum_{y : \underline{a}\cdot y = 0} (-1)^{\underline{x}_0 \cdot y} |y\rangle$$

v) Measure in comp. basis:

$\Rightarrow$ obtain random y s.th. $\underline{a} \cdot y = 0$.

$(n-1)$ lin. indep. vectors y_i (over $\mathbb{Z}_2$) s.th. $\underline{a} \cdot y_i = 0$ allow to determine $\underline{a}$ (solve lin. eq. – e.g. Gaussian elimination).

Space of lin. dep. vectors of k vectors grows as 2^k

$\Rightarrow$ $O(1)$ chance to find randomly a lin. indep. vector

$\Rightarrow$ $O(n)$ random y are enough

$\Rightarrow$ $O(n)$ oracle queries are enough (on average)

Classical: 2^{cn} queries

Quantum: $c' \cdot n$ queries

$\Big\}$ exponential speed-up !

(in terms of oracle queries)

Notes: • We don't have to measure B — we never use the outcome! (But: Derivation easier this way!)

• $H^{\otimes n} \mathrel{\hat{=}}$ (discrete) Fourier transform over $\mathbb{Z}_2^{\times n}$

$\longrightarrow$ period finding via Fourier transform

3. The quantum Fourier transform, period finding, and Shor's factoring algorithm

Can we go beyond Fourier trafo on $\mathbb{Z}_2$ (to $\mathbb{Z}_N$, for $N \sim 2^n$)?

- What is the right transformation?
- Can it be implemented efficiently?
- What is it good for?

Further reading:
A. Ekert and R. Jozsa,
Quantum computation and Shor's factoring algorithm.
Rev. Mod. Phys **68**, 733 (1996)
https:///doi.org/10.1103/RevModPhys.68.733

a) The Quantum Fourier Transform

Discrete Fourier trafo (FT) on $\mathbb{C}^N$:

$$x = (x_0, \dots, x_{N-1}) \in \mathbb{C}^N$$

$$y = (y_0, \dots, y_{N-1}) \in \mathbb{C}^N$$

FT: $\mathcal{F}: x \longmapsto y$ s.th. $y_k = \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j \, e^{2\pi i \, jk/N}$

Definition QFT

$$\boxed{|j\rangle \longmapsto \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} e^{2\pi i \, jk/N} |k\rangle}$$

Observe:

$$\sum_j x_j |j\rangle \xmapsto{QFT} \sum_{jk} x_j \, e^{2\pi i jk/N} |k\rangle = \sum y_k |k\rangle$$

i.e.: QFT acts as discrete FT on amplitudes!

Computational cost of classical FT:

- $O(N^2)$ operations.
- $N \sim 2^n \Rightarrow$ exponential in # of bits in N.
- Fast FT (FFT): only $O(N \log N)$, but still exponential!
- $O(N)$ is lower bound: minimal time to even just output y_k!

Will see: QFT can be implemented on a quantum state in $O(n^2)$ steps

$\rightarrow$ exponential speedup!

(But only useful if input is give as q. state!)

Step I: Rewrite QFT in binary

- Consider case $N = 2^n$.
- Write j etc. in binary:

$$j = j_1 j_2 j_3 \cdots j_n = j_1 \cdot 2^{n-1} + j_2 2^{n-2} + \dots + j_n 2^0$$

- "Decimal" point notation:

$$0.j_\ell j_{\ell+1} \cdots j_n = \frac{1}{2} j_\ell + \frac{1}{4} j_{\ell+1} + \dots + \frac{1}{2^{n-\ell+1}} j_n$$

Then:

$$|j\rangle \longmapsto \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j \cdot k/2^n} |k\rangle$$

$= 0.k_1 k_2 \dots k_n$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j \left(\sum_{\ell=1}^{n} k_\ell 2^{-\ell}\right)} |k_1, \dots, k_n\rangle$$

$$= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \left[\bigotimes_{\ell=1}^{n} \left(e^{2\pi i j k_\ell 2^{-\ell}} |k_\ell\rangle \right) \right]$$

$$= \bigotimes_{\ell=1}^{n} \left[\frac{1}{\sqrt{2}} \sum_{k_\ell=0}^{1} e^{2\pi i j k_\ell 2^{-\ell}} |k_\ell\rangle \right]$$

$$= \bigotimes_{e=1}^{n} \frac{1}{\sqrt{2}} \left[|0\rangle + e^{2\pi i j 2^{-e}} |1\rangle \right] = \dots$$

$$j \cdot 2^{-e} = \underbrace{j_1 j_2 \cdots j_{n-e}}_{\text{integer}} \cdot j_{n-e+1} \cdots j_n$$

$$e^{2\pi i (j \cdot 2^{-e})} = e^{2\pi i \cdot (\text{integer} + 0.j_{n-e+1} \cdots j_n)}$$

$$= e^{2\pi i \cdot 0.j_{n-e+1} \cdots j_n}$$

$$\dots = \frac{|0\rangle + e^{2\pi i\, 0.j_n} |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i\, 0.j_{n-1} j_n} |1\rangle}{\sqrt{2}} \otimes \dots$$

$$\dots \otimes \frac{|0\rangle + e^{2\pi i\, 0.j_1 j_2 \cdots j_n} |1\rangle}{\sqrt{2}}$$

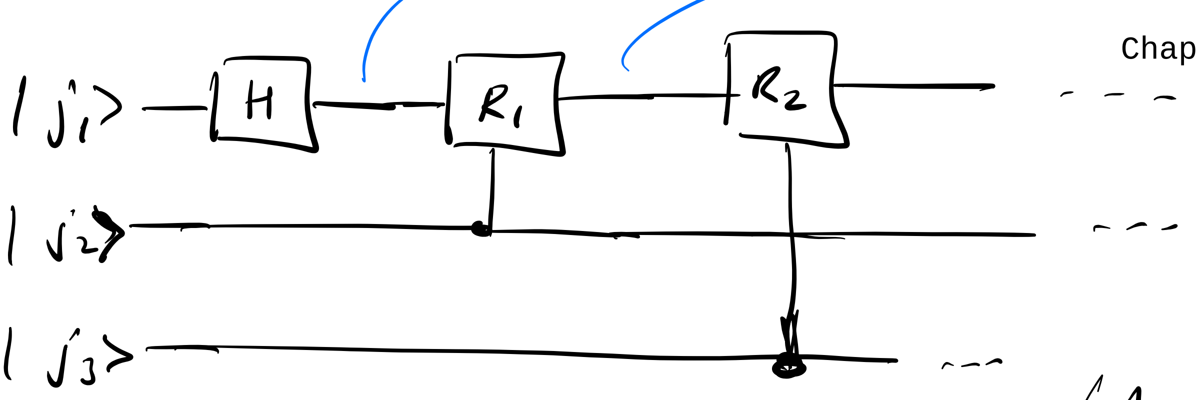
Step II: Implement this as a circuit.

Consider first only rightmost term:

$$\frac{|0\rangle + e^{2\pi i\, 0.j_1 j_2 \cdots j_n} |1\rangle}{\sqrt{2}} = \frac{|0\rangle + e^{2\pi i j_1/2} e^{2\pi i j_2/4} \cdot e^{2\pi i j_3/8} \dots |1\rangle}{\sqrt{2}}$$

$(-1)^{j_1}$

$$|0\rangle + e^{2\pi i j_1/2} |1\rangle \qquad |0\rangle + e^{2\pi i j_1/2} e^{2\pi i j_2/4} |1\rangle$$



$$R_d = \begin{pmatrix} 1 & \\ & e^{2\pi i \cdot 2^{-(d+1)}} \end{pmatrix}$$

<u>Action of gates:</u>

$$H: |j_1\rangle \longmapsto |0\rangle + e^{2\pi i\, 0.j_1}|1\rangle$$

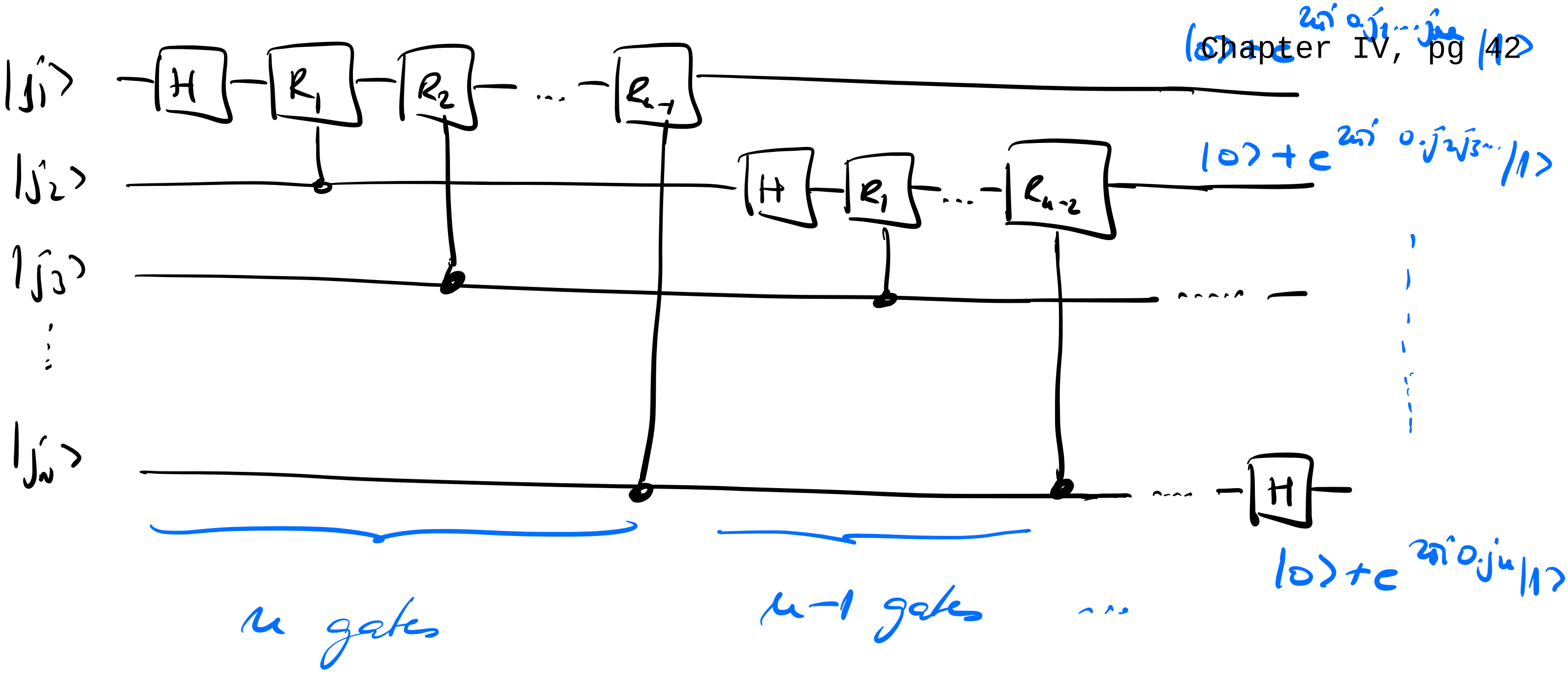
$$C\text{-}R_1: \left(|0\rangle + e^{2\pi i\, 0.j_1}|1\rangle\right)|j_2\rangle \longmapsto \left(|0\rangle + e^{2\pi i\, 0.j_1 j_2}|1\rangle\right)|j_2\rangle$$

$$C\text{-}R_2: \left(|0\rangle + e^{2\pi i\, 0.j_1 j_2}\right)|j_2\rangle|j_3\rangle \longmapsto \left(|0\rangle + e^{2\pi i\, 0.j_1 j_2 j_3}|1\rangle\right)|j_2\rangle|j_3\rangle$$

$\vdots$ and so on.

$\longrightarrow$ Outputs the n-th qubit of the QFT on 1st qubit.

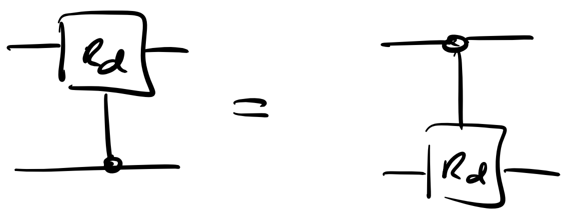
Continue on this vein:



<u>Gate count:</u> $\frac{n(n+1)}{2} = \underline{O(n^2) \text{ gates}}!$

<u>Notes:</u>

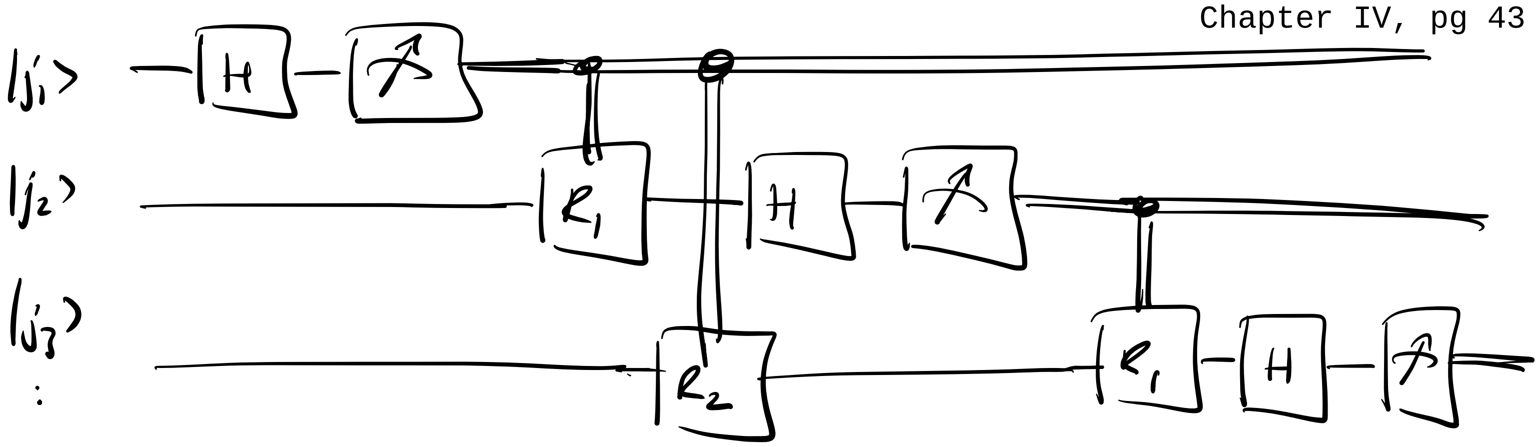
- Output qubits in <u>reverse order</u> (can re-order if needed: $n/2$ swaps).

- 

$\Longrightarrow$ can flip C-R_d gates

Then, upper line acts as <u>control</u> in comp. basis.

$\Longrightarrow$ If we <u>measure</u> directly after QFT in comp. basis, we can measure <u>before</u> the C-R_d gates & control them classically:



Only one-qubit gates needed (!!)

("Where is the quantum-ness?")

b) Period finding

Application of QFT: Find period of a function?
(cf. Simon's algorithm)

Consider a periodic function $f: \mathbb{N} \rightarrow \{0, \dots, M-1\}$, such that $\exists r > 0$ with

$$f(x) = f(x+r), \text{ and } f(x) \neq f(y) \text{ otherwise.}$$

On a computer, we can only compute f on a truncated input,

$$f: \underbrace{\{0, \dots, N-1\}}_{= \{0,1\}^n} \longrightarrow \underbrace{\{0, \dots, M-1\}}_{= \{0,1\}^m}$$

(In particular, the periodicity of f is broken across the boundary, if we think of $f(x+r) \equiv f((x+r) \bmod N)$)

Can we find r better than classically ?

(i.e., with much less than $\sim r$ queries to f)

Choose n such that $2^n \gg r$

will make this specific later. Goal: imperfection at end. negligible.

Note: Since we do not know r, we need to know some upper bound on r – e.g., we can use that $r \leq N$.

Implement U_f on quantum computer as before:

$$U_f : |x\rangle_A |y\rangle_B \longmapsto |x\rangle_A |y \oplus f(x)\rangle_B$$

<u>Algorithm:</u>

① Hadamard on A, then U_f:

$$\frac{1}{2^{n/2}} \sum |x\rangle_A |0\rangle_B \xmapsto{U_f} \frac{1}{2^{n/2}} \sum |x\rangle_A |f(x)\rangle_B$$

② Measure B register. For result $|f(x_0)\rangle_B$,

A collapses to

$$\frac{1}{\sqrt{k_0}} \sum_{k=0}^{k_0-1} |x_0 + kr\rangle$$

– here, $0 \le x_0 < r$, and $\frac{2^n}{r} - 1 < k_0 \le \frac{2^n}{r}$.

③ Apply QFT:

$$\longmapsto \frac{1}{2^{n/2}\sqrt{k_0}} \sum_{k=0}^{k_0-1} \sum_{\ell=0}^{2^n-1} e^{2\pi i (x_0 + kr)\ell/2^n} |\ell\rangle_A$$

$$= \sum_{\ell=0}^{2^n-1} \underbrace{e^{2\pi i x_0 \ell/2^n} \underbrace{\sum_{k=0}^{k_0-1} \frac{1}{2^{n/2}\sqrt{k_0}} e^{2\pi i k r \ell/2^n}}_{=: \hat{a}_\ell}}_{=: a_\ell} |\ell\rangle_A$$

④ Measure in computational basis:

$|\hat{a}_\ell|^2$: probability to obtain outcome ℓ

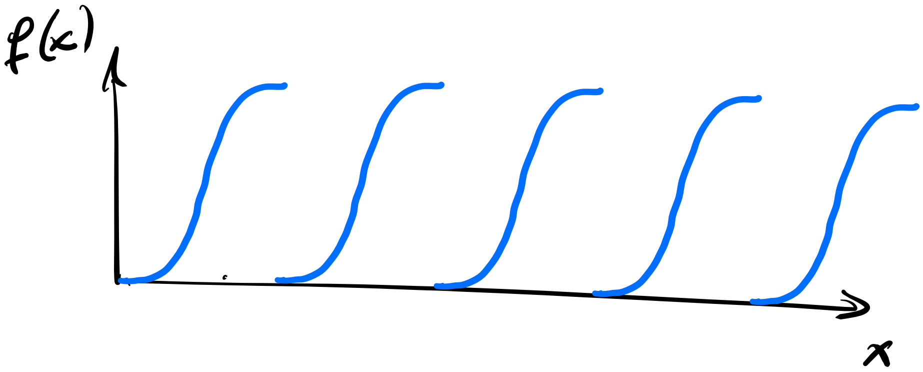
<u>Intuitively:</u> $\hat{a}_\ell \propto \sum_k e^{2\pi i k \left(r\ell/2^n\right)}$

peaked around points ℓ where $\frac{r\ell}{2^n}$ is close to an integer!

($\rightarrow$ will quantify this in a moment!)

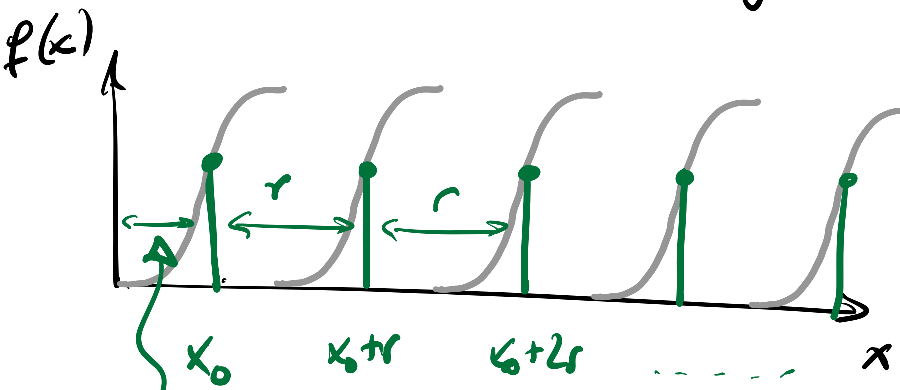
Intuitive picture:

(General features of Fourier transforms — nothing quantum!)



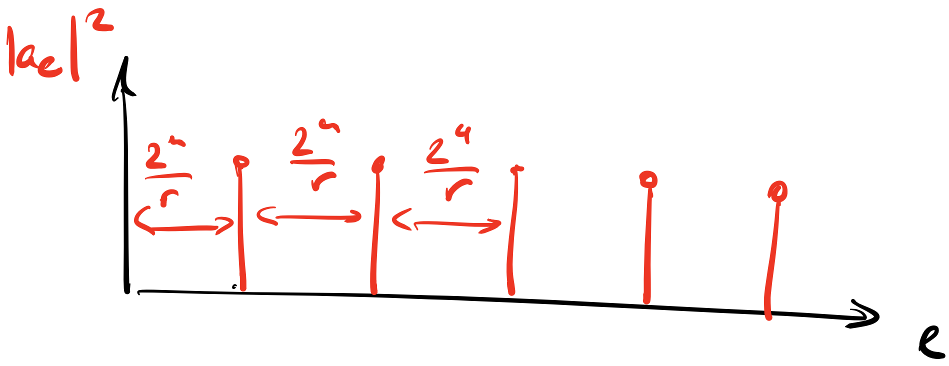
periodic function

after meas. of $B \longrightarrow x_0$



unknown offset x_0 !

Fourier trafo



unknown offset: absorbed in phase of a_ℓ !

$\Longrightarrow$ can determine multiple of $\frac{2^u}{r}$ by measuring ℓ (How to get r? Later!)

Detailed analysis of $|a_\ell|^2$:

How much total weight is on all $|a_\ell|^2$ with

$$\ell = \frac{2^u}{r}\cdot s + \delta_s \; ; \quad \delta_s \in \left(-\tfrac{1}{2}; \tfrac{1}{2}\right] ; \quad s=0,\dots,r-1$$

(i.e. only those ℓ which are closest to $\frac{2^u}{r}\cdot s$)

Then, $$\hat{a}_\ell = \frac{1}{2^{u/2}\sqrt{k_0}} \sum_{k=0}^{k_0-1} e^{2\pi i k\overbrace{\left(s+\frac{r}{2^u}\delta_s\right)}^{\equiv r\ell/2^u}}$$

$$= \frac{1}{2^{u/2}\sqrt{k_0}} \; \frac{e^{2\pi i \frac{r}{2^u}\delta_s k_0} - 1}{e^{2\pi i \frac{r}{2^u}\delta_s} - 1}$$

... since $\frac{2^u}{r} - 1 < k_0 \leq \frac{2^u}{r}$, and $r \ll 2^u$:

$$\frac{k_0 r}{2^u} = 1 - \varepsilon \,, \quad 0 \leq \varepsilon < \frac{r}{2^u} \ll 1 .$$

$$= \frac{1}{2^{n/2}\sqrt{k_0}} \; \frac{e^{2\pi i \delta_s (1-\varepsilon)} - 1}{e^{2\pi i \frac{r}{2^n}\delta_s} - 1}$$

$$\Rightarrow \quad |a_e|^2 = \frac{1}{2^n k_0} \left(\frac{\overbrace{|\sin(\pi \delta_s (1-\varepsilon))|}^{|\sin x| \geq \frac{|x|}{\pi/2} \text{ on relev. interval}}}{\underbrace{|\sin(\frac{\pi r}{2^n}\delta_s)|}_{|\sin x| \leq |x|}} \right)^2$$

$$\geq \frac{1}{\cancel{2^n} k_0} \; \frac{\frac{\cancel{\pi^2} \cancel{\delta_s^2} (1-\varepsilon)^2}{\pi^2/4}}{\frac{\cancel{\pi^2} r^2}{(2^n)^{\cancel{2}}} \cancel{\delta_s^2}}$$

$$= \frac{4}{\pi^2} \frac{1}{r} \; \frac{(1-\varepsilon)^2}{\underbrace{\frac{k_0 r}{2^n}}_{\geq 1-\varepsilon}}$$

$$= \frac{4}{\pi^2} \frac{1}{r} (1-\varepsilon) \approx \frac{4}{\pi^2} \frac{1}{r}$$

(can be easily made more quantitative, using $\varepsilon < \frac{r}{2^n}$!)

since $s = 0, \dots, r-1$: Total probability that

$\left| l - \frac{2^n}{r} s \right| \leq \frac{1}{2}$ for one such s: $P \geq \frac{4}{\pi^2} \approx 0.41$

With sufficiently high probability — we will see that we can <u>check</u> success and thus repeat until we succeed! — we obtain an l s.th. $l = \frac{2^n}{r} s + \delta_s$, and thus,

$$\frac{l}{2^n} \approx \frac{s}{r},$$

where s is chosen uniformly at random.

If we choose $r \ll 2^n$ suitably, there is only <u>one</u> such ratio $\frac{s}{r}$ with $\left| l - \frac{2^n}{r} s \right| \leq 1/2$, and it can be found efficiently. (See further reading.)

Specifically, it suffices to choose $N = 2^n = (2^m)^2 = M^2$, i.e. $n = 2m$, and since $M \geq r$: $2^n \gg 2^{n/2} > r$.

If s and r are co-prime, i.e. $\gcd(r,s)=1$, we can infer r from $\frac{s}{r}$. This happens with probability at least $p(\gcd(s,r)=1) \geq 1/\log r \geq \frac{2}{\log 2}\cdot\frac{1}{n}$.

(at least all primes $2 \leq s < r$ are good, and density of primes goes as $1/\log r$.)

$\Longrightarrow$ with $O(n)$ iterations, we find a s coprime with r.

Once we have used this to obtain a guess for r, we can test whether $f(x) = f(x+r)$, and repeat until success!

$\Longrightarrow$ Efficient algorithm for period finding.

$\sim O(n)$ applications of f required!

c) Application: Factoring Algorithm

Factoring: Given $N \in \mathbb{N}$ (not prime), find

$f \in \mathbb{N}$, $f \neq 1$, such that $f \mid N$.

"f divides N"

(Note: Primality of N can be checked efficiently.)

This can be solved efficiently if we have an efficient method for period finding!

Sketch of algorithm:

(1) Select a random a, $2 \leq a < N$.

If $\underbrace{\gcd(a,N)}_{\text{ef. computable!}} > 1 \implies$ done, $f = \gcd(a,N)$!

Thus: Assume $\gcd(a,N) = 1$.

(2) Denote by r the smallest $x > 0$ such that

$$a^x \bmod N = 1.$$

– that is, the period of

$$f_{N,a}(x) := a^x \bmod N$$

r is called the <u>order of a mod N.</u>

(<u>Note</u>: Some $z \geq 1$ s.th. $a^z \bmod N = 1$ must exist

since

$\exists\, x, y \in \{1, \dots, N\}: \; a^x \equiv a^y \bmod N$ (counting possibilities)

$\Rightarrow a^x\left(1 - a^{y-x}\right) \equiv 0 \bmod N$

$\Rightarrow N \mid \left(a^x\left(1 - a^{y-x}\right)\right)$

$\overset{\gcd(a,N)=1}{\Longrightarrow} N \mid \left(1 - a^{y-x}\right)$

$\Rightarrow a^{y-x} \equiv 1 \bmod N$ ∎)

Recall: "<u>Efficient</u>" means "<u>polynomial in # of digits of N</u>"

Furthermore, $f_{N,a}(x)$ can be <u>computed efficiently</u>:

Using $x = x_{m-1} 2^{m-1} + x_{m-2} 2^{m-2} + \dots$,

$$a^x \bmod N = \underbrace{\left(a^{(2^{m-1})}\right)}\,^{x_{m-1}} \cdot \left(a^{(2^{m-2})}\right)^{x_{m-2}} \cdot \ldots \bmod N$$

eff. computable via repeated squaring mod N:

$$a \mapsto a^2 \bmod N \mapsto \underbrace{a^4 \bmod N}_{\equiv (a^2 \bmod N)^2 \bmod N} \mapsto \ldots,$$

by doing "mod N" in each step the numbers don't require an exp. number of digits:

$O(n)$ multiplications of n-digit numbers.

$\Rightarrow$ r can be found efficiently with a quantum computer!

③ Assume for now r even:

$$a^r \bmod N = 1$$

$$\Leftrightarrow N \mid (a^r - 1),$$

$$\Leftrightarrow N \mid (a^{r/2}+1)(a^{r/2}-1)$$

However, we also know that $N \nmid (a^{r/2} - 1)$,

since otherwise $a^{r/2} \bmod N = 1$ ↯ does not divide

$\Longrightarrow$ <u>either</u> $N \mid a^{r/2} + 1$

<u>or</u> N has non-trivial common factors with <u>both</u> $a^{r/2} \pm 1$.

$$\Longrightarrow \quad 1 \neq f := \gcd(N, a^{r/2}+1) \mid N$$

$\Longrightarrow$ found a <u>non-trivial factor f of N!</u>

$\Longrightarrow$ Algorithm will succeed as long as

(i) r even

(ii) $N \nmid (a^{r/2}+1)$

This can be shown to happen with prob. $\geq 1/2$ for a random choice of a (see further reading)

– <u>unless</u> either N is even (can be checked efficiently),

<u>or</u> $N = p^k$, p prime

(can also be checked efficiently by taking roots; there are only $O(\log(N))$ roots which one has to check!)

- <u>and</u> in <u>both cases</u>, this gives a non-trivial factor!

$\Rightarrow$ <u>efficient Quantum Algorithm for Factoring.</u>

<u>"Shor's algorithm"</u>

4. Grover's algorithm

For many hard computational problems, it is possible to check a solution efficiently, but we don't know how to find it.

So-called "NP problems" (non-deterministic polynomial)

Many interesting problems are of this type:

- graph coloring
- factoring
- 3-SAT
- tiling problems
- Hamiltonian path
- travelling salesman (suitably phrased)

⋮

Reformulation of NP problems:

We have an efficiently computable function

$$f(x) \in \{0, 1\} \; ; \quad x \in \{0, 1, \dots, N-1\}$$

efficient =

— where $f(x)$ is a "verifier" for a given "solution" x, i.e. $f(x)=1$ if & only if x is a correct solution —

and we want to find an x_0 s.th. $f(x_0)=1$ (i.e. a correct solution).

(Can be interpreted as "database search": Want to find a "marked element" x_0 in an unstructured database.)

We assume for now that $x_0 : f(x_0)=1$ is unique.

(Generalization: later/homework)

Classically: Will need $O(N)$ queries to f for an unstructured search (i.e., without using properties of f).

Quantumly: Will see that $O(\sqrt{N})$ queries are enough.

(<u>Note</u>: This is only a <u>quadratic</u> speedup, but it applies to a <u>very large class</u> of very relevant problems.)

Consider $f: \{0, \dots, N-1\} \to \{0,1\}$

$N = 2^n$: n qubits

<u>Ingredient 1</u>:

Oracle

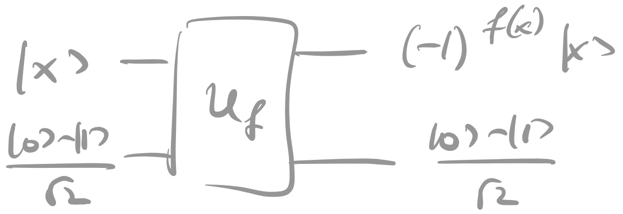
$$O_f : |x\rangle \longmapsto (-1)^{f(x)} |x\rangle = (-1)^{\delta_{x,x_0}} |x\rangle$$

i.e. O_f flips amplitude of "marked" element.

Can also write as

$$\boxed{O_f = I - 2|x_0\rangle\langle x_0|}$$

(<u>Note</u>: Can build O_f from U_f via phase kick-back technique:

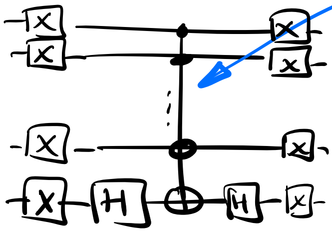


)

Ingredient 2:

Unitary $O_0 : |x\rangle \longmapsto (-1)^{\delta_{x,0}} |x\rangle$

n-qubit Toffoli!

Corresponds to  , or O_f, $f(x) = \delta_{x,0}$

$\longrightarrow$ can be realized efficiently.

Again, write $O_0 = I - 2|0\rangle\langle 0|$

Define

$$\boxed{O_\omega := H^{\otimes n} O_0 H^{\otimes n} = I - 2|\omega\rangle\langle\omega|},$$

with $|\omega\rangle = \frac{1}{\sqrt{N}} \sum_{x=0}^{N-1} |x\rangle$.

Grover's algorithm:

① Start from $|\psi_0\rangle = |\omega\rangle = H^{\otimes n}|0\rangle$

② Repeat: Apply Grover iteration

$$G = -H^{\otimes n} O_0 H^{\otimes n} O_f = (-O_\omega) O_f$$

$$|\psi_k\rangle \xmapsto{G} |\psi_{k+1}\rangle = G|\psi_k\rangle = (-O_\omega) O_f |\psi_k\rangle.$$

(How many times? — Soon!)

How to analyze "trajectory" $|\psi_0\rangle \to |\psi_1\rangle \to |\psi_2\rangle \to \dots$?

<u>Recall</u>: $O_f = I - 2|x_0\rangle\langle x_0|$

$$-O_\omega = 2|\omega\rangle\langle\omega| - I$$

and moreover, $|\psi_0\rangle = |\omega\rangle$.

$\Rightarrow$ Only two <u>special vectors</u> in $|\psi_0\rangle$, O_f, $-O_\omega$: $|x_0\rangle$ and $|\omega\rangle$. The identity I will not change these vectors.

$\Rightarrow$ The dynamics $|\psi_0\rangle \to |\psi_1\rangle \to |\psi_2\rangle \to \dots$ takes place in span$\{|x_0\rangle, |\omega\rangle\}$, i.e., a two-dimensional space!

Two natural ONBs for Alex's space:

$$|x_0\rangle$$

$$|x_0^\perp\rangle := \frac{1}{\sqrt{N-1}} \sum_{x \neq x_0} |x\rangle \qquad \langle x_0 | x_0^\perp\rangle = 0$$

$$\propto |\omega\rangle - |x_0\rangle\underbrace{\langle x_0|\omega\rangle}_{= 1/\sqrt{N}}$$

And another basis

$$|\omega\rangle$$

$$|\omega^\perp\rangle \qquad \text{with} \quad \langle \omega | \omega^\perp \rangle = 0$$

Of course, any vector in span$\{|x_0\rangle, |\omega\rangle\}$ can be expanded in either basis:

$$|\psi\rangle = a|x_0\rangle + b|x_0^\perp\rangle = x|\omega\rangle + y|\omega^\perp\rangle$$

What is the effect of O_f and $(-O_\omega)$ on $|\psi\rangle$?

$$O_f|\psi\rangle = O_f\left(a|x_0\rangle + b|x_0^\perp\rangle\right) = -a|x_0\rangle + b|x_0^\perp\rangle$$

$$O_f = I - 2|x_0\rangle\langle x_0|$$

$\Rightarrow$ O_f reflects $|\psi\rangle$ about $|x_0^\perp\rangle$!

$$(-O_\omega)|\psi\rangle = (-O_\omega)\left(x|\omega\rangle + y|\omega^\perp\rangle\right)$$

$$\overset{O_\omega = I - 2|\omega\rangle\langle\omega|}{=} -\left(-x|\omega\rangle + y|\omega^\perp\rangle\right) = x|\omega\rangle - y|\omega^\perp\rangle.$$

$\Rightarrow$ $(-O_\omega)$ reflects $|\psi\rangle$ about $|\omega\rangle$!

Thus: each Grover iteration consists of two steps:

(i) reflect about $|x_0^\perp\rangle$

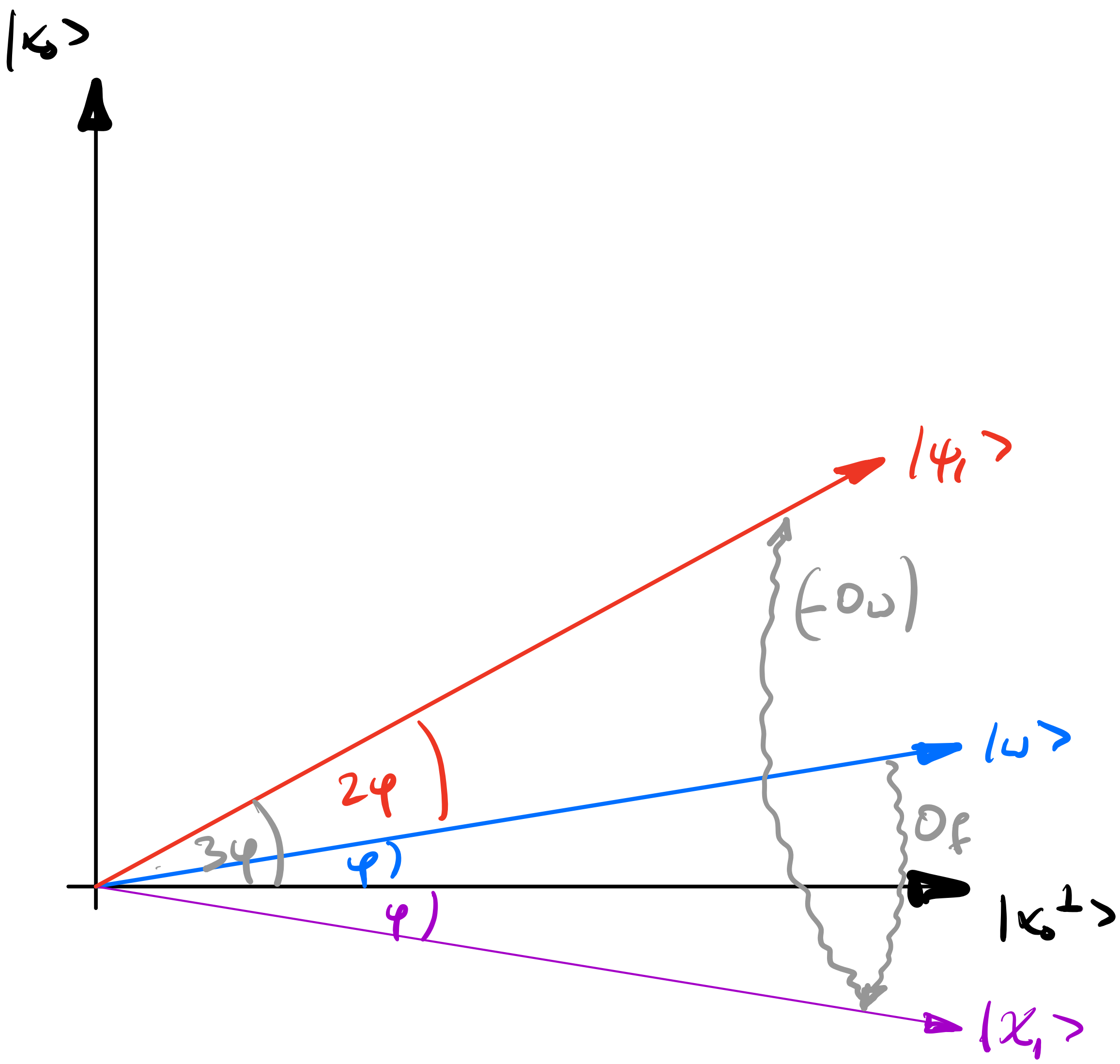
(ii) reflect about $|\omega\rangle$

What happens now if we start with $|\psi_0\rangle = |\omega\rangle$ and apply one iteration?

$$|\omega\rangle = \sin\varphi\,|x_0\rangle + \cos\varphi\,|x_0^\perp\rangle.$$

$$|\chi_1\rangle = O_f\,|\omega\rangle$$

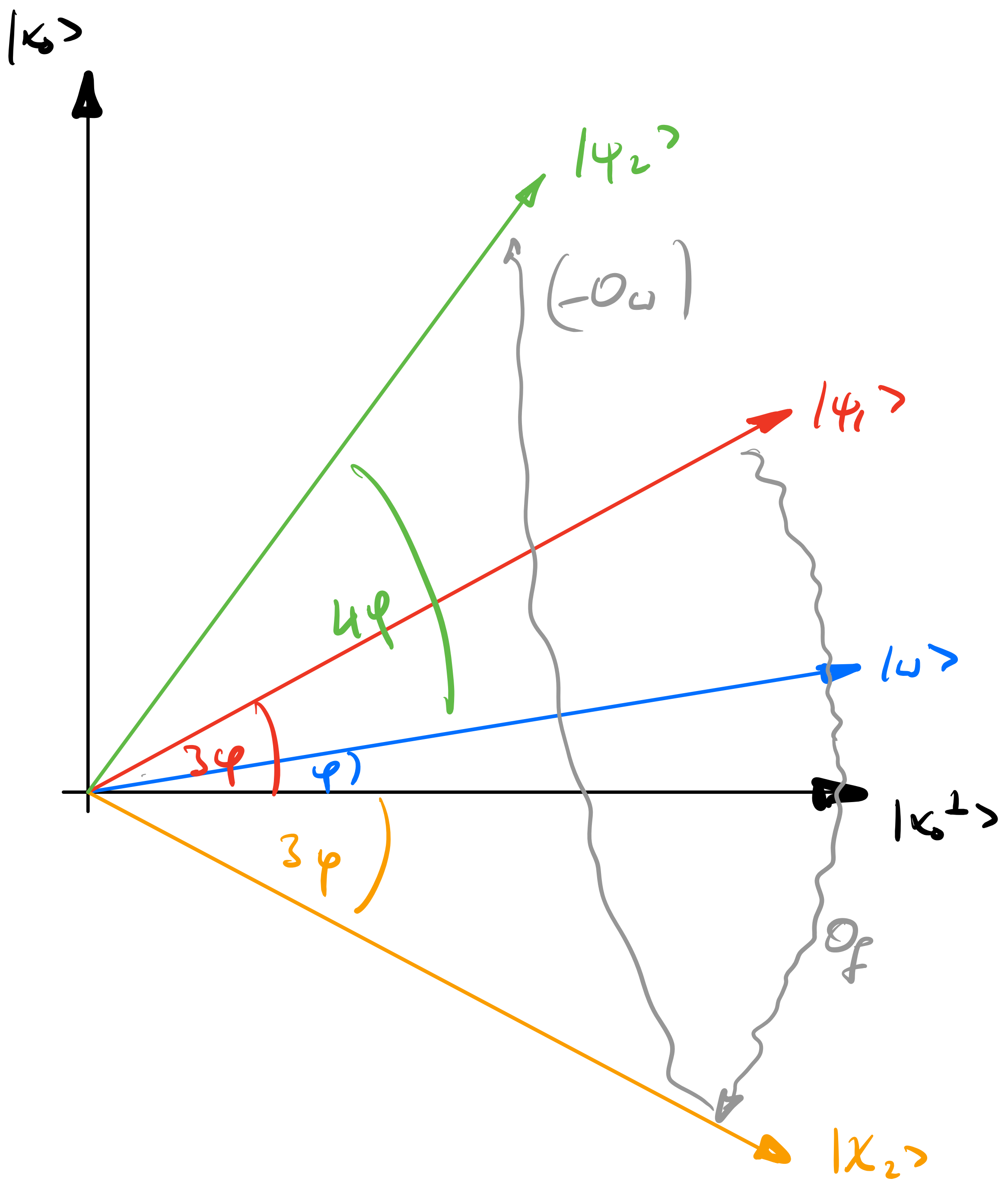
$$|\psi_1\rangle = (-O_\omega)|\chi_1\rangle = (-O_\omega)\,O_f\,|\omega\rangle$$



$$\Rightarrow \quad |\psi_1\rangle = \sin(3\varphi)\,|\kappa_0\rangle + \cos(3\varphi)\,|\kappa_0^\perp\rangle$$

Next Grover iteration:

$$|\psi_2\rangle = (-O_\omega)\,\underbrace{O_f\,|\psi_1\rangle}_{=:\,|\chi_2\rangle}$$



$$\Rightarrow |\psi_2\rangle = \sin(5\varphi)\,|x_0\rangle + \cos(5\varphi)\,|x_0^\perp\rangle$$

Can continue ... !

$$\Rightarrow |\psi_k\rangle = \sin\big((2k+1)\varphi\big)|x_0\rangle + \cos\big((2k+1)\varphi\big)|x_0^\perp\rangle$$

Want that $(2k+1)\varphi \approx \frac{\pi}{2}$: Then, measurement in comp. basis will return $|x_0\rangle$ with high prob.!

Since $$|\omega\rangle = \frac{1}{\sqrt{N}}|x_0\rangle + \sqrt{\frac{N-1}{N}}\,|x_0^\perp\rangle$$

$$= \sin\varphi\,|x_0\rangle + \cos\varphi\,|x_0^\perp\rangle$$

$$\Rightarrow \quad \frac{\sin\varphi}{\cos\varphi} = \frac{\sqrt{\frac{1}{N}}}{\sqrt{\frac{N-1}{N}}} = \frac{1}{\sqrt{N-1}}$$

$\Rightarrow$ for large N, $\varphi \approx \frac{1}{\sqrt{N}}$.

$\Rightarrow$ Need $k \approx \frac{\pi}{4}\cdot\sqrt{N}$ Grover iterations.

$\Rightarrow$ $O(\sqrt{N})$ calls to f (for O_f) sufficient!

<u>Quadratic speed-up</u> with respect to classical algorithms for <u>general search problems</u>!

Note:

- If there are $K > 1$ solutions:

  Same method with $O\left(\sqrt{\frac{N}{K}}\right)$ steps works

  ($\rightarrow$ homework)

- Can also be adapted to case where number of solutions is unknown.