Norbert Schuch

QUANTUM COMPUTING

and

QUANTUM ALGORITHMS

Wintersemester 2024

Transcribed and Typeset by
Maximilian Fettinger

University of Vienna

1 The circuit model

1.1 Classical computation

Use of classical computers (abstractly):

Solve problems ≡ compute functions:

$$f : \{0,1\}^n \to \{0,1\}^m$$
$$\underline{x} = (x_1, \ldots, x_n) \mapsto f(x_1, \ldots, x_n)$$

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

Example. Problem = **multiplication**: $(a, b) \mapsto a \cdot b$

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

Problem = **factorization**:

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

More precisely: Each problem is encoded by a **family** of functions $f \equiv f^{(n)} : \{0,1\}^n \mapsto \{0,1\}^m$, with $m = \underbrace{\mathrm{poly}(n)}$, $n \in \mathbb{N}$ – one for each input size.

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

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

Which ingredients do we need to compute a **general function** f?

(i)

$$f : \{0,1\}^n \to \{0,1\}^m$$
$$f(\underline{x}) = (f_1(\underline{x}), \ldots, f_m(\underline{x}))$$

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

$\Longrightarrow$ can restrict analysis to boolean functions

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

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

Define

$$\delta_{\underline{y}}(\underline{x}) = \begin{cases} 0 & \text{if } \underline{x} \neq \underline{y} \\ 1 & \text{if } \underline{x} = \underline{y} \end{cases}$$

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

(iii) Define **bitwise** δ:

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

Then

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

(iv)

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

Combine (i) - (iv): Any $f(\underline{x})$ can be constructed from **4 ingredients**: AND, OR and NOT gates, plus a COPY gate $x \mapsto (x, x)$.

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 $f \equiv f^{(n)}$ which we can compute are construced by **concatening 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 problems**

- $K(n) \gg \text{poly}(n)$ – e.g. $K(n) \sim \exp(n^\alpha)$: **hard problems**

Note (Technical). We must suppose that the circuits used for $f^{(n)}$ are **uniform**, 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.

Example. f = Multiplication:

Efficient:

$$
\begin{array}{l}
\underline{\overbrace{10110}^{l}\times\overbrace{10011}^{l'}} \\
10110 \\
\quad 10110 \\
\underline{\quad\;\; 10110} \\
\underbrace{110100010}_{l\times l' \text{ additions: } O(ll')\sim O(n^2) \text{ gates}}
\end{array}
$$

f = Factorization:

E.g.: Sieve of Eratosthenes:

$$\{0,1\}^n \to \text{ try about } \sqrt{2^n} \sim 2^{\frac{n}{2}} \text{ cases}$$

$\Longrightarrow$ hard/exponential scaling

No efficient algorithm known!

Is a **typical problem easy or hard**?

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

Number of different f: $2^{(2^n)}$

But: there are only $c^{\text{poly}(n)}$ circuits of length poly(n) – with c denoting the number of elementary gates.

$\Longrightarrow$ 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!

Remark. 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 ...

But: All known *reasonable* models of computation can simulate each other with poly(n) overhead $\Longrightarrow$ same computational power (in the sense above).

Church-Turing thesis: All reasonable models of computation have the same computational power.

1.2 Reversible circuits

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

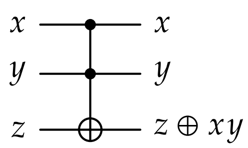
Gates will be replace by **unitaries**

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**:

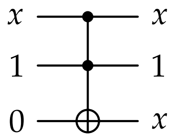
Toffoli gate:



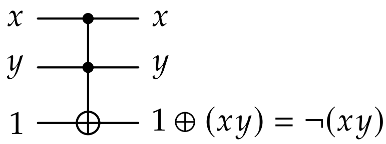
$\rightarrow$ Toffoli gate is reversible (it is its own reverse, since $(z \oplus xy) \oplus xy = z$)

$\rightarrow$ Toffoli gate can simulate AND/OR/NOT/COPY, by using ancillas in state *0* or *1*:

E.g.:



COPY



NAND

$\Longrightarrow$ 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 number of gates:

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

Idea. Replace any gate by a reversible gate using ancillas. Then *XOR* the result into $\underline{y}$ register. Finally, run the circuit backwards to *uncompute* the ancillas. Ancilla count can be optimised for $\rightarrow$ cf. Preskill's notes

$\Longrightarrow$ Everything can be **computed reversibly**.

BUT: **3-bit** gate is **required**!

1.3 Quantum Circuits

Most common model for quantum computation:

The circuit model:

- Quantum system consisting of qubits: tensor product structure
- **Universal gate set** $S = \{U_1, \ldots U_k\}$ of few-qubit gates (typically 1- and 2-qubit gates) U_j. (See later for definition of *universal*!)
- Construct circuits by sequentially applying elements of S to a subset of qubits:
$$|\psi_{\text{out}}\rangle = V_T \ldots V_2 V_1 |\psi_{\text{in}}\rangle$$
where V_j are U_j acting on a subset of qubits.

- Initial state:

$$
\begin{aligned}
|\psi_{\text{in}}\rangle &= |x_1\rangle |x_2\rangle \dots |x_n\rangle \overbrace{|0\rangle |0\rangle \dots |0\rangle}^{l} \\
&= \underbrace{|\underline{x}\rangle}_{\text{encodes instance of problem}} \underbrace{|\underline{0}\rangle}_{\text{ancillas}}
\end{aligned}
$$

  - alternatively, we can also have

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

    and encode the instance in the circuit.

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

  $\longrightarrow$ outcome $|\underline{y}\rangle$ with probability $p(y) = |\langle \underline{y}|\psi_{\text{out}}\rangle|^2$

Notes:

- This is a **probabilistic** scheme – it outputs $\underline{y}$ with some probability $p(\underline{y})$. In principle, we should compare to **classical probabilistic** schemes – see later.
- We need not measure all qubits – not measuring = trancing = 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 measurement outcome: This dependence can be realized **inside** the circuit with *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 realized **exactly**. $\longrightarrow$ Requires a **continuous family** of universal gates (counting argument!).
  - **approximate universality**: Any n-qubit gate can be approximated well be gate set (finite gate set sufficient;

Solovan-Kitaev theorem: ε-approximation (in $\|\cdot\|_\infty$-Norm) of 1-qubit gate requires $O(\text{poly}(\log(1/\varepsilon)))$ 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** (with probability 1) ***single*** **two-qubit gate** will do!
- More universal sets: later!

1.4 Universal gate set

Our **exact universal gate set**:

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

$$R_x(\phi) = e^{-i\phi\frac{X}{2}};\ X = \begin{pmatrix} 0 & 1 \\ 1 & 0 \end{pmatrix},\ X^2 = I$$

$$R_z(\phi) = e^{-i\phi\frac{Z}{2}};\ Z = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix},\ Z^2 = I$$

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

$$\Longrightarrow R_x(\phi) = \begin{pmatrix} \cos\frac{\phi}{2} & -i\sin\frac{\phi}{2} \\ -i\sin\frac{\phi}{2} & \cos\frac{\phi}{2} \end{pmatrix}$$

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

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

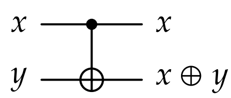
Together, R_x and R_z generate all rotations in SO(3) (Euler angles!), and thus in SU(2) up to a phase.

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!

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

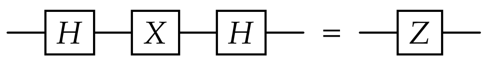
Overview of a number of important gates and identities

Hadamard gate :

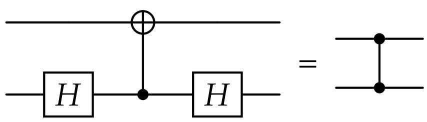
$$H = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}; \quad H = H^\dagger; \quad H^2 = I.$$

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

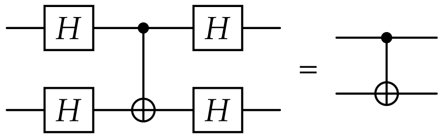
Graphical *circuit* notation:



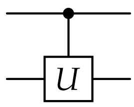
Important. Matrix notation: time goes right to left.
Circuit notation: time goes left to right.

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

controlled-Z, *controlled-Phase*, CZ, CPHASE



Generally: For a unitary $U \in \mathrm{SU}(2)$

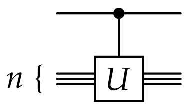


$$= \left(\begin{array}{c|c} I & 0 \\ \hline 0 & U \end{array}\right)$$

controlled-U

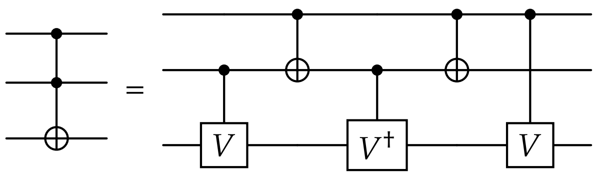
Can be implemented with 2 CNOT (homework).

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



$$= \left(\begin{array}{c|c} I_{2^n} & 0 \\ \hline 0 & U \end{array}\right)$$

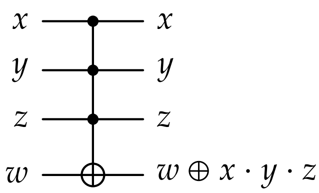
Circuit for Toffoli:



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 with 3 controls: can be built from normal Toffoli (since classically universal!)

Finally, some **further approximate universal gate sets**:

- CNOT + 2 random 1-qubit gates
- CNOT + H + $T = R_z(\frac{\pi}{4})$ ($\frac{\pi}{8}$ *gate*)

2 Oracle-based algorithms

2.1 The Deutsch algorithm

Consider $f : \{0,1\}^n \to \{0,1\}$. 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?

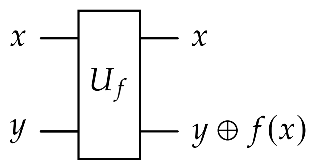
Classically, we clearly need **2 queries**:

$$\text{Compute } f(0) \text{ and } f(1).$$

Can quantum physics help?

Consider **reversible implementation** of f:

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

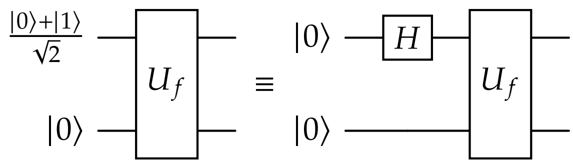


$$|x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$$

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 superposition as inputs**?

First attempt:



$$\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes |0\rangle = \frac{1}{\sqrt{2}}(|0\rangle |0\rangle + |1\rangle |0\rangle) \overset{U_f}{\longmapsto} \boxed{\frac{1}{\sqrt{2}}(|0\rangle |f(0)\rangle + |1\rangle |f(1)\rangle)}$$

$\longrightarrow$ Have evaluated f on **both outputs**!

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

- Measure in computational basis: **collapse** superposition to **one case**!
- More generally: If $\boldsymbol{f(0) \neq f(1)}$, the output is in

  $$S_{\neq} = \left\{1/\sqrt{2}(|00\rangle + |11\rangle), 1/\sqrt{2}(|01\rangle + |10\rangle)\right\},$$

  and for $\boldsymbol{f(0) = f(1)}$ in

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

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

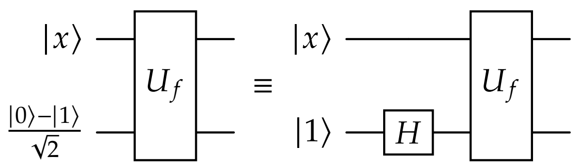
  But: We can do measurements which, with some probability, allows to conclude that $f(0) = f(1)$ or $f(0) \neq f(1)$. E.g., all states in $S_=$ are orthogonal to $R_{\neq} = \{|+\rangle|0\rangle, |+\rangle|1\rangle\}$, and all states in $S_{\neq}$ to $R_= = \left\{\frac{|00\rangle-|11\rangle}{\sqrt{2}}, \frac{|01\rangle-|10\rangle}{\sqrt{2}}\right\}$.

  $\Longrightarrow$ A **POVM** 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: $\frac{1}{2}$ (homework)

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

Second attempt:



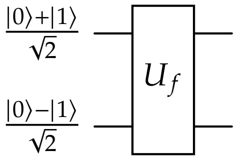
$$|x\rangle\left(\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \xmapsto{U_f} |x\rangle\left(\frac{|f(x)\rangle - |1 \oplus f(x)\rangle}{\sqrt{2}}\right)$$

$$= \begin{cases} |x\rangle\left(\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right) & \text{if } f(x) = 0 \\ |x\rangle\left(\frac{|1\rangle-|0\rangle}{\sqrt{2}}\right) & \text{if } f(x) = 1 \end{cases}$$

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

Not useful by itself: $f(x)$ only encoded in **global phase** for each **classical input** $|x\rangle$

Combine attempts:



:

$$
\begin{aligned}
\frac{|0\rangle + |1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}} &= \frac{1}{\sqrt{2}}\left(|0\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} + |1\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&\xmapsto{U_f} \frac{1}{\sqrt{2}}\left((-1)^{f(0)}|0\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}} + (-1)^{f(1)}|1\rangle \frac{|0\rangle - |1\rangle}{\sqrt{2}}\right) \\
&= \frac{(-1)^{f(0)}|0\rangle + (-1)^{f(1)}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle - |1\rangle}{\sqrt{2}}
\end{aligned}
$$

Observations:

→ No entanglement created!

→ 2nd qubit – the one where U_f outputs the fuction value – is unchanged!!

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

(phase kick-back technique)

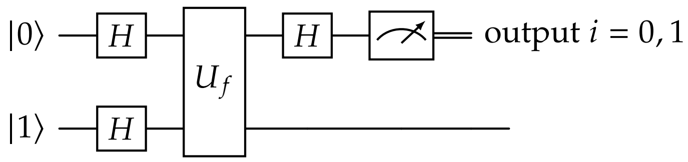
State of 1st qubit:

$$
\begin{aligned}
f(0) = f(1) &\iff \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\
f(0) \neq f(1) &\iff \frac{|0\rangle - |1\rangle}{\sqrt{2}}
\end{aligned}
$$

(up to irrelevant global phase)

Orthogonal states! $\implies$ measurement of 1st qubit in basis $\{|+\rangle, |-\rangle\}$ (or apply $-\boxed{H}-$ and measure in computational basis) allows to decide if $f(0) \stackrel{?}{=} f(1)$!

Deutsch algorithm:



$$\begin{aligned} \text{output } i = 0 &\implies f(0) = f(1) \\ i = 1 &\implies f(0) \neq f(1) \end{aligned}$$

One application of U_f has been sufficient!

$\implies$ Speed-up compared to **classical algorithm** (1 vs. 2 oracle queries).

Interesting to note: 2nd qubit never needs to be measured – and it contains **no information**.

Two main insights:

- Use input $\sum |x\rangle$ to **evaluate f on all inputs simultaneously**.
- This parallelism alone is not enough – **need a smart way to read out the relevant information**.

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

Thus:

2.2 The Deutsch-Josza algorithm

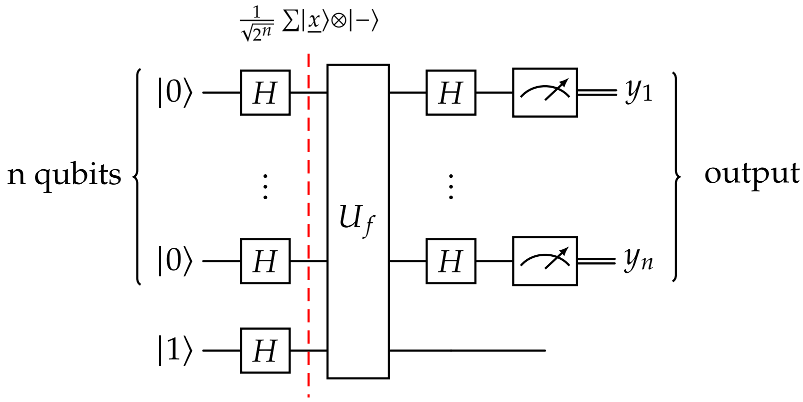
Consider $f : \{0,1\}^n \to \{0,1\}$ with **promise** (i.e., a condition we know is met by f) that

$$\begin{aligned} &\textbf{either } f(\underline{x}) = c \quad \forall \underline{x} && \textit{(f constant)} \\ &\textbf{or } |\{\, \underline{x} \mid f(\underline{x}) = 0 \,\}| = |\{\, \underline{x} \mid f(\underline{x}) = 1 \,\}| && \textit{(f balanced)} \end{aligned}$$

Want to know: Is f **constant** or **balanced**?

How many queries are needed?

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



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

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

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

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

$$\textbf{or:} \quad |\underline{x}\rangle \mapsto \frac{1}{\sqrt{2^n}} \sum_{\underline{y}} (-1)^{\underline{x}\cdot\underline{y}}\, |\underline{y}\rangle$$

where $\underline{x} \cdot \underline{y} = x_1 y_1 \oplus \ldots \oplus x_n y_n$

("scalar product" mod 2, **NOT** a scalar product!)

Analysis of circuit: (we omit normalization)

$$
\begin{aligned}
|\underline{0}\rangle\,|1\rangle &\xmapsto{H^{\otimes n}\otimes H} \left(\sum_{\underline{x}} |\underline{x}\rangle\right)(|0\rangle - |1\rangle) \\
&\xmapsto{U_f} \left(\sum_{\underline{x}} (-1)^{f(\underline{x})}\,|\underline{x}\rangle\right)(|0\rangle - |1\rangle) \\
&\xmapsto{H^{\otimes n}\otimes I} \left(\sum_{\underline{y}} \underbrace{\sum_{\underline{x}} (-1)^{f(\underline{x})+\underline{x}\cdot\underline{y}}}_{=:a_{\underline{y}}}\,|\underline{y}\rangle\right)(|0\rangle - |1\rangle)
\end{aligned}
$$

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

***f* constant:** $f(\underline{x}) = c$

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

***f* balanced:** $|\{\,\underline{x} \mid f(\underline{x}) = 0\,\}| = |\{\,\underline{x} \mid f(\underline{x}) = 1\,\}|$

For $\underline{y} = \underline{0}$:

$$
a_{\underline{0}} = \sum_{\underline{x}} (-1)^{f(\underline{x})+\underline{x}\cdot\underline{0}} = \sum_{\underline{x}} (-1)^{f(\underline{x})} \underset{f \text{ balanced}}{=} 0
$$

Thus:

$$
\begin{aligned}
\text{Output } \underline{y} = \underline{0} &\Longrightarrow f \text{ constant} \\
\text{Output } \underline{y} \neq \underline{0} &\Longrightarrow f \text{ balanced}
\end{aligned}
$$

$\Longrightarrow$ We can **unambiguously distinguish** between the 2 cases with **one query to the oracle for *f***!

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! $\Longrightarrow$ **exponential vs. constant**!
- **But:** If we are ok to get right answer with very high proability $p = 1-p_{\text{error}}$, then for k queries to f,
$$p_{\text{error}} \approx \underbrace{\frac{1}{2^k}}_{\approx \text{ probability to get } k \text{ times the same result for balanced } f, \text{ if } k \ll 2^n}$$
i.e.: $k \sim \log(1/p_{\text{error}})$.
- **Randomized classical**: Much smaller speed-up vs. randomized classical algorithms (even for exponentially small error, $k \sim n$ oracle calls are sufficient.)

2.3 Simon's algorithm

... will give us a true exponential speed-up (also relative to **randomized** classical algorithms) in terms of oracle queries.

Oracle: $f : \{0,1\}^n \to \{0,1\}^n$ with the **promise**:
$$\exists \underline{a} \neq \underline{0} \text{ s.th. } f(\underline{x}) = f(\underline{y}) \textbf{ exactly if } \underline{y} = \underline{x} \oplus \underline{a}.$$
(hidden periodicity)

Task: Find $\underline{a}$ by querying f.

Classical: 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 $x_1, \dots, x_k \to \sim k^2$ pairs, for each pair: $\Pr(f(\underline{x}_i) = f(\underline{x}_j)) \approx 2^{-n}$

$\Longrightarrow p_{\text{success}} \sim k^2 2^{-n} \Longrightarrow$ need $k \sim 2^{n/2}$ 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 $U_f : |x\rangle |y\rangle \mapsto |x\rangle |y \oplus f(x)\rangle$
$$\left(\frac{1}{\sqrt{2^n}} \sum_{\underline{x}} |\underline{x}\rangle_A\right) |0\rangle_B \overset{U_f}{\longmapsto} \frac{1}{\sqrt{2^n}} \sum_{\underline{x}} |\underline{x}\rangle_A |f(\underline{x})\rangle_B$$

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

$\Longrightarrow$ Register A colapses onto

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

How can we **extract $\boldsymbol{a}$**?
(Measure in computational basis $\rightarrow$ collapse on random $\underline{x}_0$: useless)

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

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

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

(v) Measure in **computational basis**:

$\Longrightarrow$ obtain **random $\underline{\boldsymbol{y}}$** s.th. $\underline{a}\cdot\underline{y} = 0$.

$(n-1)$ linear independent vectors $\underline{y}_i$ (over $\mathbb{Z}_2$) s.th. $\underline{a}\cdot\underline{y}_i = 0$ allow to determine $\underline{a}$. (solve linear system of equations – e.g. Gaussian elimination).

Space of linear dependent vectors of k vectors grows as 2^k.
$\Longrightarrow \mathcal{O}(1)$ chance to find randomly linear independent vector
$\Longrightarrow \mathcal{O}(n)$ random $\underline{y}$ are enough
$\Longrightarrow$ **$\boldsymbol{\mathcal{O}(n)}$ oracle queries** are enough (on average)

$$\left.\begin{array}{l}\textbf{Classical: } 2^{cn} \text{ queries}\\ \textbf{Quantum: } c'n \text{ queries}\end{array}\right\} \textbf{exponential speed-up!}$$

Notes:

- We don't have to measure B – we never use the outcome! (But: Derivation easier this way!)
- $H^{\otimes n} \triangleq$ (discrete) Fourier transform on $\mathbb{Z}_2^{\times n} \longrightarrow$ **period finding** via **Fourier transform**.

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

Can we go beyond Fourier transform 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?

3.1 The quantum Fourier transform

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

$$\begin{aligned} x &= (x_0, \ldots, x_{N-1}) \in \mathbb{C}^N \\ y &= (y_0, \ldots, y_{N-1}) \in \mathbb{C}^N \\ FT : \mathcal{F} : \; x \mapsto y \; \text{ s.th } \; y_k &= \frac{1}{\sqrt{N}} \sum_{j=0}^{N-1} x_j e^{2\pi i jk/N} \end{aligned}$$

Definition (Quantum Fourier transform (QFT))**.**

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

Observe:

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

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

Computational cost of classical FT:

- $O(N^2)$ operations.
- $N \sim 2^n \Longrightarrow$ exponential in number of bits in N.
- Fast Fourier transform (FFT): only $O(N \log N)$ operations, bit **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.

$\Longrightarrow$ exponential speed-up!

(But only useful if input is given as quantum state!)

Step I: Rewrite QFT in binary

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

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

- *Decimal* point notation:

$$0.j_l j_{l+1} \ldots j_n = \frac{1}{2} j_l + \frac{1}{4} j_{l+1} + \ldots + \frac{1}{2^{n-l+1}} j_n$$

Then

$$\begin{aligned}
|j\rangle \mapsto \frac{1}{2^{n/2}} \sum_{k=0}^{2^n-1} e^{2\pi i j \overbrace{k/2^n}^{=0.k_1k_2\ldots k_n}} |k\rangle
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} e^{2\pi i j (\sum_{l=1}^{n} k_l 2^{-l})} |k_1, k_2, \ldots, k_n\rangle \\
&= \frac{1}{2^{n/2}} \sum_{k_1=0}^{1} \cdots \sum_{k_n=0}^{1} \left[\bigotimes_{l=1}^{n} (e^{2\pi i j k_l 2^{-l}} |k_l\rangle) \right] \\
&= \bigotimes_{l=1}^{n} \left[\frac{1}{\sqrt{2}} \sum_{k_l=0}^{1} e^{2\pi i j k_l 2^{-l}} |k_l\rangle \right] \\
&= \bigotimes_{l=1}^{n} \frac{1}{\sqrt{2}} \left[|0\rangle + e^{2\pi i j 2^{-l}} |1\rangle \right]
\end{aligned}$$

Auxiliary calculation:

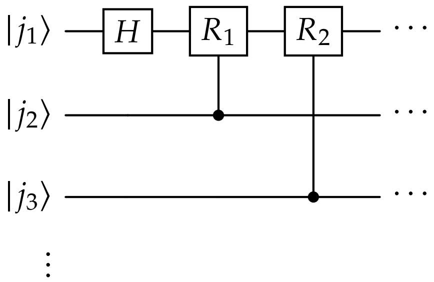
$$j \cdot 2^{-l} = \underbrace{j_1 j_2 \ldots j_{n-l}}_{\text{integer}} . j_{n-l+1} \ldots j_n$$

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

$$\dots = \frac{|0\rangle + e^{2\pi i 0.j_n}|1\rangle}{\sqrt{2}} \otimes \frac{|0\rangle + e^{2\pi i 0.j_{n-1}j_n}|1\rangle}{\sqrt{2}} \otimes \dots \otimes \frac{|0\rangle + e^{2\pi i 0.j_1 j_2 \dots j_n}|1\rangle}{\sqrt{2}}$$

Step II: Implement this **as a circuit**. Consider first only **rightmost term**:

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



$$R_d = \begin{pmatrix} 1 & 0 \\ 0 & e^{2\pi i \cdot 2^{-(d+1)}} \end{pmatrix}$$

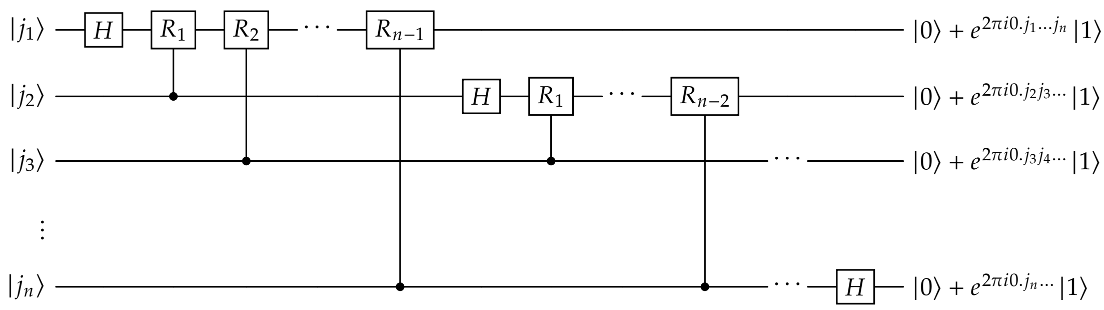
Actions of gates:

$$\begin{aligned}
H : |j_1\rangle \quad & \mapsto |0\rangle + e^{2\pi i 0.j_1}|1\rangle \\
\text{C-}R_1 : \left(|0\rangle + e^{2\pi i 0.j_1}|1\rangle\right)|j_2\rangle \quad & \mapsto \left(|0\rangle + e^{2\pi i 0.j_1 j_2}|1\rangle\right)|j_2\rangle \\
\text{C-}R_2 : \left(|0\rangle + e^{2\pi i 0.j_1 j_2}|1\rangle\right)|j_2\rangle|j_3\rangle \quad & \mapsto \left(|0\rangle + e^{2\pi i 0.j_1 j_2 j_3}|1\rangle\right)|j_2\rangle|j_3\rangle
\end{aligned}$$

and so on ...

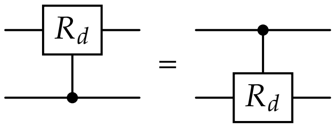
$\longrightarrow$ Outputs the n-th qubit of the QFT on 1st qubit.

Continue in this vein:



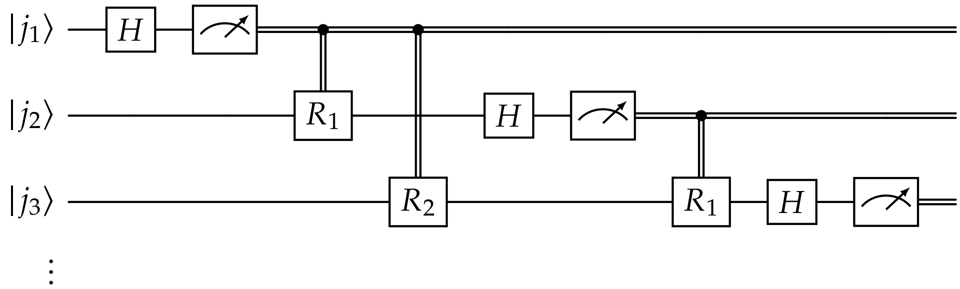
Gate count: $\frac{n(n+1)}{2} = \boldsymbol{O(n^2)}$ **gates!**

Notes:

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

$\Longrightarrow$ can flip C-R_d gates

  Then, **upper** line acts as **control** in computational basis.
  $\Longrightarrow$ If we **measure** directly after QFT in computational basis, we can measure **before** the C-R_d gates and control them classically:



Only **one-qubit gates** needed!!

(*Where is the quantum-ness?*)

3.2 Period finding

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

Consider a periodic function $f : \mathbb{N} \to \{0, \ldots, M-1\}$ such that $\exists r > 0$ with $f(x) = f(x+r)$, and $f(x) \neq f(y)$ otherwise.

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

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

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

Can we find r better than classically? (i.e., with much less than $\sim r$ queries to f)

Chosse n such that $2^n \underbrace{\gg}_{\text{will make this specific later}} r$

Note. Since we do not know r, we need to know some upper bound on r – e.g., we can use that $r < M$.

Implement U_f in quantum computer as before:

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

Algorithm:

(1) Hadamard on A, then U_f:

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

(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:

$$\begin{aligned} &\mapsto \frac{1}{2^{n/2}\sqrt{k_0}} \sum_{k=0}^{k_0-1} \sum_{l=0}^{2^n-1} e^{2\pi i (x_0+kr)l/2^n} |l\rangle_A \\ &= \sum_{l=0}^{2^n-1} \underbrace{e^{2\pi i x_0 l/2^n} \underbrace{\sum_{k=0}^{k_0-1} \frac{1}{2^{n/2}\sqrt{k_0}} e^{2\pi i r k l/2^n}}_{=:\hat{a}_l}}_{=:a_l} |l\rangle_A \end{aligned}$$

(4) Measure in computational basis:

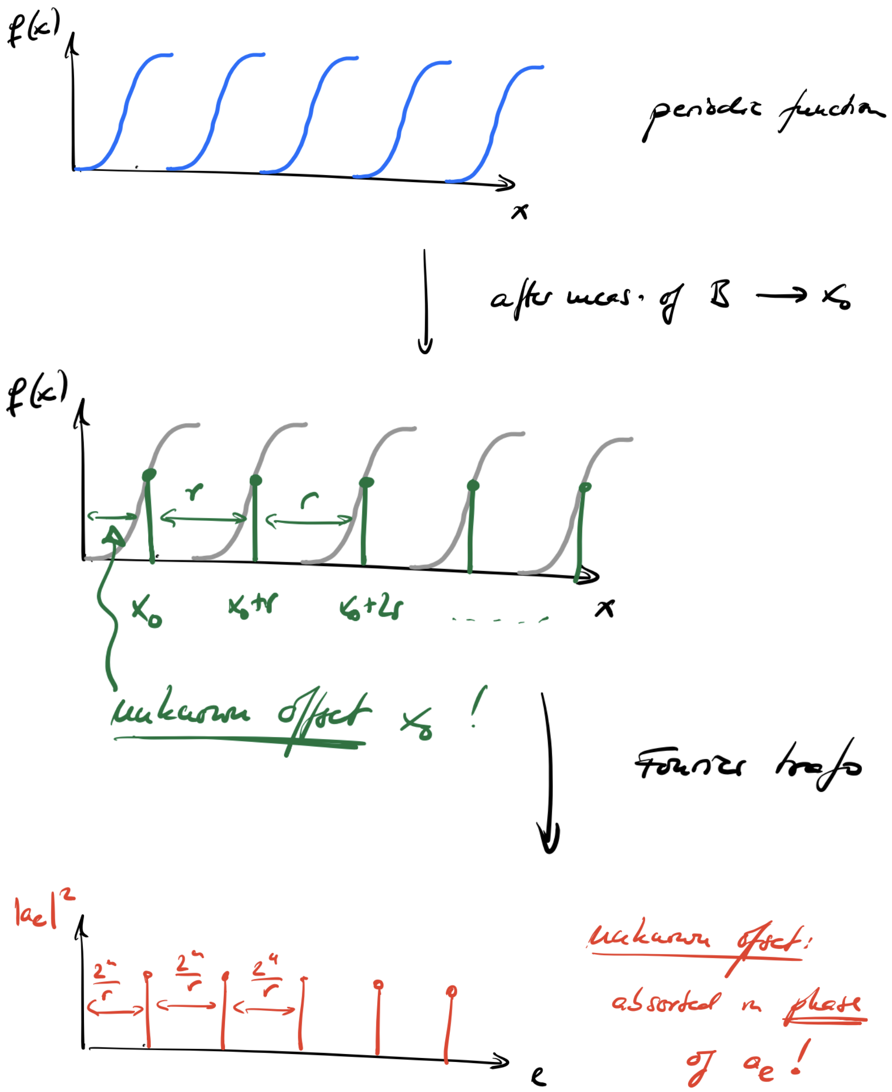
$$|\hat{a}_l|^2 : \text{ probability to obtain outcome } l$$

Intuitively: $\hat{a}_l \propto \sum_k e^{2\pi i \left(\frac{rkl}{2^n}\right)}$ peaked around points l where $\frac{rl}{2^n}$ is a close to an integer!

($\longrightarrow$ Will quantify this in a moment!)

Intuitive picture:

(General features of Fourier transforms – nothing quantum!)



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

Detailed analysis of $|a_l|^2$: How much **total weight** is in all $|a_l|^2$ with

$$l = \frac{2^n}{r} \cdot s + \delta + s; \ \delta_s \in \left(-\frac{1}{2}, \frac{1}{2}\right]; \ s = 0, 1, \ldots, r-1$$

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

$$\text{Then, } \hat{a}_l = \frac{1}{2^{n/2}\sqrt{k_0}} \sum_{k=0}^{k_0-1} e^{2\pi i k \overbrace{(2 + \frac{r}{2^n}\delta_s)}^{\equiv rl/2^n}}$$

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

$$\boxed{\ldots \text{since } \frac{2^n}{r} - 1 < k_0 \leq \frac{2^n}{r}, \text{ and } r \ll 2^n : \frac{k_0 r}{2^n} = 1 - \varepsilon,\ 0 \leq \varepsilon \ll 1.}$$

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

$$\Longrightarrow |\hat{a}_l|^2 = \frac{1}{2^n k_0} \Bigg(\frac{\overbrace{|\sin(\pi\delta_s(1-\varepsilon))|}^{|\sin x| \geq \frac{|x|}{\pi/2} \text{ in relevant interval}}}{\underbrace{|\sin(\frac{\pi r}{2^n}\delta_s)|}_{|\sin x| \leq |x|}} \Bigg)^2 \geq \frac{1}{\cancel{2^n} k_0} \frac{\frac{\cancel{\pi^2}\cancel{\delta_s^2}(1-\varepsilon)}{\pi^2/4}}{\frac{\cancel{\pi^2} r^2}{(2^n)^{\cancel{2}}}\cancel{\delta_s^2}}$$

$$= \frac{4}{\pi^2}\frac{1}{r}\frac{(1-\varepsilon)^2}{\frac{k_0 r}{2n}} = \frac{4}{\pi^2}\frac{1}{r}(1-\varepsilon) \approx \frac{4}{\pi^2}\frac{1}{r}$$

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

Since $s = 0, 1, \ldots, r-1$: Total probability that $|l - \frac{2^n}{r}s| \leq \frac{1}{2}$ for one such s: $p \geq \frac{4}{\pi^2} \approx 0.41$

With sufficiently high probability – we will see that we can **check** 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 **one** such ratio $\frac{s}{r}$ with $|l - \frac{2^n}{r}s| \leq \frac{1}{2}$, and it can be found efficiently. (see further reading.)

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

If s and r are coprime, i.e. $\gcd(r,s) = 1$, we can infer r from $\frac{s}{r}$. This happens with probability at least $\Pr(\gcd(s,r) = 1) \geq \frac{1}{\log r} \geq \frac{2}{\log 2} \cdot \frac{1}{n}$. (at least all primes $2 \leq s < r$ are good, and density of primes is $\frac{1}{\log r}$)

$\Longrightarrow$ 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; $O(n)$ applications of f required!

3.3 Application: Factoring algorithm

Factoring: Given $N \in \mathbb{N}$ (not prime), find $f \in \mathbb{N}$, $f \neq 1$, such that $f|N$.

(*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, $2 \leq a < N$.

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

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

(2) Denote by r the smallest $x > 0$ such that $a^x \mod N = 1$. – that is, the period of $f_{N,a}(x) := a^x \mod N$.

r is called the **order of a mod N**.

(*Note:* Some $z > 1$ s.th. $a^z \mod N = 1$ must exist since

$$\begin{aligned} \exists x, y \in \{1, \ldots, N\} :\ & a^x \equiv a^y \mod N \text{ (counting possibilities)} \\ & \Longrightarrow a^x(1 - a^{y-x}) \equiv 0 \mod N \Longrightarrow N|(a^x(1 - a^{y-x})) \\ & \xRightarrow{\gcd(a,N)=1} N|(1 - a^{y-x}) \Longrightarrow a^{y-x} \equiv 1 \mod N \qquad \square\,) \end{aligned}$$

Furthermore, $f_{N,a}(x)$ can be **computed efficiently** (recall: *Efficient* means *polynomial in number of digits of* N):

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

$$a^x \mod N = \underbrace{(a^{(2^{m-1})})}^{x_{m-1}} \cdot (a^{(2^{m-2})})^{x_{m-2}} \cdot \ldots \mod N$$

Effeciently computable via reapeated squaring mod N:
$a \mapsto a^2 \mod N \mapsto a^4 \mod N \mapsto \ldots$,
by doing "mod N" at each step the numbers don't require an exponential number of digits:
$O(n)$ multiplications of n-digit numbers.

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

(3) Assume for now ***r* even**:

$$a^r \mod N = 1 \Longleftrightarrow N|(a^r - 1) \Longleftrightarrow N|(a^{r/2} + 1)(a^{r/2} - 1)$$

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

$\Longrightarrow$ **either** $N|(a^{r/2} + 1)$ **or** N has non-trivial common factors with **both** $a^{r/2} \pm 1$.

$$\Longrightarrow 1 \neq f := \gcd(N, a^{r/2} + 1) \nmid N$$
$$\Longrightarrow \text{ found a } \textbf{non-trivial factor } \boldsymbol{f} \textbf{ on } \boldsymbol{N}\textbf{!}$$

$\Longrightarrow$ Algorithm will succeed as long as

(i) r is even

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

This can be shown to happen with probability $\geq \frac{1}{2}$ for a random choice of a (see further reading).

- **unless** either N is even (can be checked efficiently) **or** $N = p^k$ for some prime p (can also be checked efficiently by taking roots; there are only $O(\log N)$ roots which one has to check!)
- **and** in **both cases**, this gives a non-trivial factor!

$\Longrightarrow$ **Efficient Quantum Algorithm for Factoring.**

Shor's Algorithm