The Complexity Classes P and NP

consider another 4 problems.

- Multiply 7 by 5 and check that it equals 35.
- Multiply 11 by 17 and check that it equals 187.
- Multiply 29 by 83 and check that it equals 2407.
- Multiply 337 by 263 and check that it equals 88,631.

Each subsequent question (bigger input size) takes more time to solve than the previous one, but the amount of time is growing more slowly.

Consider the following 4 problems.

- Find two whole numbers bigger than 1 whose product is equal to 35.
- Find two whole numbers bigger than 1 whose product is equal to 187.
- Find two whole numbers bigger than 1 whose product is equal to 2,407.
- Find two whole numbers bigger than 1 whose product is equal to 88,631.

each subsequent question (bigger input size) is harder and will take more steps and consequently more time to solve

We denote the number of digits of the input number by n, so in the first set of questions we start with $n = 2$ and go up to $n = 5$.

let $T\ (n)$ denotes the time, or equivalently the number of steps, to solve a question of input length n.

Complexity looks at how the size of $T\ (n)$ grows as n grows.

In particular, we ask if we can find some positive numbers k and p such that $T(n) \leq kn^p$ for every value of n.

If we can, we say that the underlying problem can be solved in ***polynomial time***.

If, on the other hand, we can find positive number k and a number $c > 1$, such that $T(n) > kc^n$ for every value of n, we say that the problem requires ***exponential time***.

Note: *polynomial versus exponential growth*: Given enough time, something with exponential growth will grow much faster than something with polynomial growth.

In computer science, questions that can be solved in polynomial time are considered ***easy (tractable)***, but those with exponential growth are ***hard (not tractable).***

There is a hope to solve, most polynomial time with a large value of *n* at the moment, in a few years. On the other hand, there is no hope to solve a hard problem problem, once the size has increased beyond what we can currently tackle , in the foreseeable future. So, increasing the size of *n* even slightly more, produces a problem that becomes much harder.

Back to two sets of problems.

The second set involves multiplying two numbers together, but this is easy to do. As *n* increases it does take more time, but it can be shown that this is a polynomial time problem.

What about the first set of questions?

If you tried tackling them, you will probably believe that the amount of time needed is exponential in *n* and not polynomial in *n*, but is this the case? Everybody thinks so, but, on the other hand, nobody has found a proof.

complexity class P: If a problem can be ***solved*** in ***polynomial***.

So, the problem that consists of multiplying two numbers together belongs to *P*.

complexity class NP (nondeterministic polynomial):

note: NP refers to certain types of Turing machines that are called ***nondeterministic Turing machines***

Instead of ***solving*** the problem, someone gives you the answer and you just have to ***verify*** that the answer is correct. If this process of verifying that an answer is correct takes polynomial time.

The problem of *factoring a large number* into the product of two primes belongs to *NP*.

every problem that is in P is also in NP:

since, *verifying* that an answer is correct is easier than actually *finding the answer*,

but what about the converse question. ***Does every NP problem belong to P?*** Is it true that every question whose answer can be *verified in polynomial time* can also be *solved in polynomial time*?

nobody has managed to prove that *P* is not equal to *NP*. The problem of factoring a large number into the product of two primes belongs to *NP*, and we don't think it belongs to *P*, but nobody has been able to prove it.
The problem of whether *NP* is equal to *P* is one of the most important in computer science. In 2000, the Clay Mathematics Institute listed seven "Millennium Prize Problems," each with a prize of a million dollars. The *P* versus *NP* problem is one of the seven.

A formal summary:

The class **P** consists of all those *decision problems* that can be *solved on a deterministic sequential machine* in an amount of **time that is polynomial in the size of the input**;

the class **NP** consists of all those *decision problems* whose positive solutions can be ***verified** in polynomial* time given the right information, or equivalently, whose ***solution** can be found in polynomial time on a non-deterministic machine* [wikipedia.org 2004].
Where:
† **A *decision problem*** is a function that takes an arbitrary value or values as input and returns a yes or a no. Most problems can be represented as decision problems.
† ***Witnesses*** are solutions to *decision problems* that can be checked

in polynomial time. For example, checking if 7747 has a factor less than 70 is a decision problem. 61 is a factor (61 *£* 127 = 7747) which is easily checked.
So we have a witness to a yes instance of the problem. Now, if we ask if 7747 has a factor less than 60 there is no easily checkable witnesses for the no instance [Nielsen, M. A. 2002].
† ***Non deterministic*** Turing machines (NTMs) differ from normal *deterministic*
Turing machines in that at each step of the computation the Turing machine can "spawn" copies, or new Turing machines that work in parallel with the original. *It's a common mistake to call a quantum computer an NTM, as we shall see later we can only use quantum parallelism indirectly.*
† It is not proven that **P ≠NP** it is just very unlikely as this would mean that all problems in **NP** can be solved in polynomial time.

Are Quantum Algorithms Faster Than Classical Ones?
Most *quantum computer scientists believe that P is not equal to NP*. They also ***think that there are problems that are in NP but not P, which a quantum computer can solve in polynomial time.*** This means that there are problems that a quantum computer can solve in polynomial time that a classical computer cannot.

To prove this, however, involves the first step of showing that some problem belongs to *NP* but not to *P*, and as we have seen, nobody knows how to do this.

How can we compare the speed of quantum algorithms to classical algorithms?

There are **two ways**: theoretical, and practical.

- The theoretical way is to invent a new way of measuring complexity that makes it easier to construct proofs.

- The practical way is to construct quantum algorithms for solving important real world problems in polynomial time that we believe, but have been unable to prove, do not belong to *P*.

An example of the practical approach is Shor's algorithm for factoring the product of two primes. Peter Shor constructed a quantum algorithm that works in polynomial time. We believe, but have been unable to prove, that a classical algorithm cannot do this in polynomial time

First approach:

Query Complexity

Most quantum algorithms that concern *evaluating functions*.

E.g.

- Deutsch and Deutsch-Jozsa algorithms consider functions that belong to two classes.
- Simon's algorithm concerns periodic functions of a special type. Again we are given one of these functions at random, and we have to determine the period.

To run these algorithms we have to evaluate the functions. The *function* (aka a ***black box*** or an ***oracle***).

The ***query complexity*** counts the number of times that we have to evaluate the function to get our answer.

So we are querying the black box or the oracle to evaluate the corresponding function. The point of this is that we don't have to worry about how to write an algorithm that emulates the function, so we don't have to calculate the number of steps that function takes to evaluate the input. We *just keep track of the number of questions*. This is much simpler.

Complexity Classes

In complexity theory, the main classification is between problems that take polynomial time to solve (***efficient***) and those that need more than polynomial time.

Polynomial time algorithms are regarded as being practical even for very large values of *n*, but non-polynomial time algorithms are regarded as being infeasible for large *n*.

Class P:

Problems that classical algorithms can *solve* in polynomial time.

Class QP (***EQP***, for exact quantum polynomial time):

Problems that quantum algorithms can solve in polynomial time.

Usually when we use these terms we are referring to the number of steps that an algorithm takes,

query complexity a new way of measuring complexity that counts the number of questions we need to ask an oracle. The oracle that prepares the "query" reply requires to be implemented efficiently (aka ***uniformaity***).

Problems that ***separate P and QP*** : e.g. the Deutsch-Jozsa problem is not in the class P, but belonged to QP for query complexity. (The constant function is a degree 0 polynomial.) This is sometimes described as saying that the Deutsch-Jozsa—it is a problem that belongs to QP but not to P for query complexity.

classical algorithms : worst-case analysis

Assume for Deutsch-Jozsa problem. Determine given a function that takes 10 inputs and is either balanced or constant. There are $2^{10} = 1024$ possible inputs. The worst-case scenario is when the function is balanced, but we get the same answer for the first 512 evaluations, and then on the 513th evaluation we get the other value.

How likely is this to happen?

If the function is balanced, for each input value we are equally likely to get either a 0 or a 1. This can be compared to tossing a fair coin and obtaining a head or a tail. How likely is it to toss a fair coin 512 times and get heads every time? The answer is$\left(\frac{1}{2}\right)^{512}$

***bounded-error complexity classes*.** we look at algorithms that can answer the question within our bound for error.
Assume for Deutsch-Jozsa problem, we need at least a 99.9 percent success rate, or equivalently an error rate of less than 0.1 percent.

Probability of getting 0:

If a function is balanced the probability of evaluating the function 11 times and getting ***0*** every time is $(0.5)^{11}$ =0.00049 to five decimal places.

Probability of getting 1:

Similarly, the probability of obtaining **1** every time is 0.00049.

Thus the probability of obtaining the *same answer* (*all 0 or all 1*), 11 times in a row when the function is even is just less than 0.001 (0.00049 + .00049) .

So for a bound on the probability of error less than 0.1 percent, we run at most 11 function evaluations. If during the process we get both a 0 and a 1, we stop and favors the *balanced*. If all 11 evaluations are the same, we will say the function is *constant*. We could be wrong, but our error rate is less than our chosen bound.

This argument works for any *n*. In every case, we need 11 function evaluations at most.

class BPP (bounded-error probabilistic polynomial time)*Problems*: *problems* that the *classical algorithms* can ***solve*** in polynomial time with the *probability of error* within some *bound*

The Deutsch-Jozsa problem is in the class *BPP*.
Note: Is a problem could be in *BPP* for one bound on the probability of error, but not in the class *BPP* for a smaller bound? No

If the problem is in the class *BPP*, it will be there for every choice of the bound.

Class BQP (bounded-error quantum polynomial time) problems: Problems that *quantum algorithms* can *solve* in polynomial time with the probability of error within some bound

Simon's algorithm shows the problem belongs to *BQP* for query complexity (thus Simon's algorithm is not in class *QP*)

Simon: we need to keep sending qubits through the circuit until we have $n-1$ linearly independent equations. in the worst case this process can go on forever.

So, probabilistically, we may calculate N so that $\left(\frac{1}{2}\right)^N$ is less than a given error bound.

It can be shown that if we run the circuit $n+N$ times, the probability of the $n+N$ equations containing a system $n-1$ linearly independent equations is greater than $1-\left(\frac{1}{2}\right)^N$.

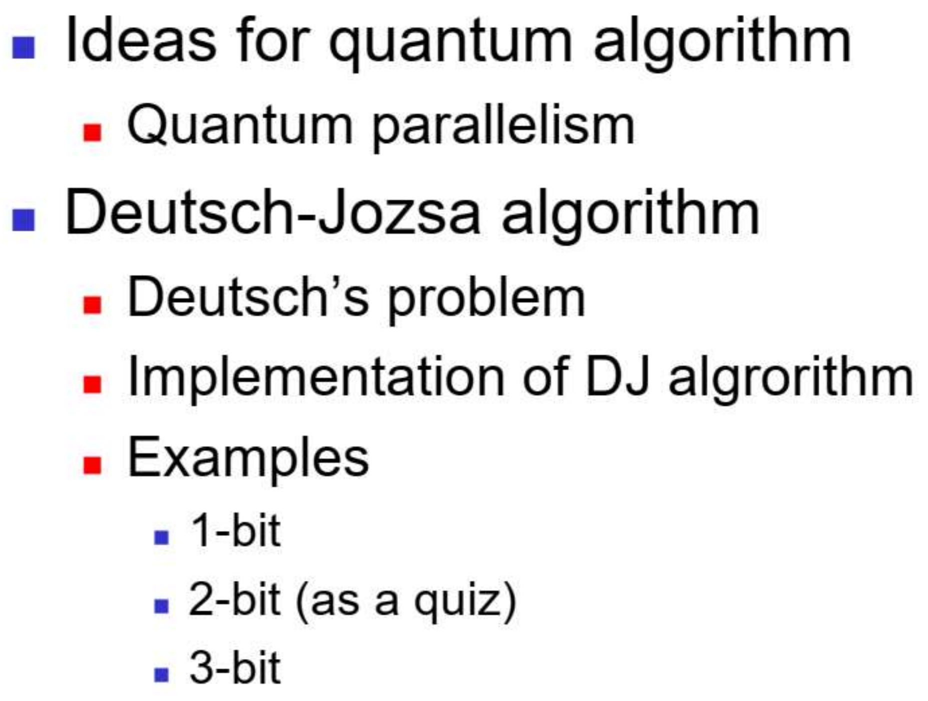
Simon's algorithm synopsis:

- First we decide on a bound on the probability of error and calculate the value N. Again, the number N does not depend on n. We can use the same value of N in each case.

- We run Simon's circuit $n + N$ times. The number of queries is $n + N$, which, since N is fixed, is a linear function of n. (We make the assumption that our system of $n + N$ equations contains $n - 1$ independent vectors. We could be wrong, but the probability of being wrong is less than the bound that we chose.)
- Then we solve the system of $n + N$ equations using a classical algorithm. The time taken will be *quadratic* in $n + N$, but because N is a constant, this can be expressed as a *quadratic* in ***n***.
  The algorithm as a whole contains the quantum part that takes linear time added to the classical part that takes quadratic time, giving quadratic time overall.

*Simon's problem **separates** BPP and BQP for query complexity:*

We showed that the classical algorithm (deterministic), in the worst case, took $2^{n-1} + 1$ function evaluations—this is exponential in n, not polynomial, so the problem definitely does not belong to *P*. It can also be shown that classical algorithm (probabilistic), even if we allow a bound on the probability of error the algorithm is still exponential, so the problem does not belong to *BPP.*
Deutsch-Jozsa (nielsen book page 36)



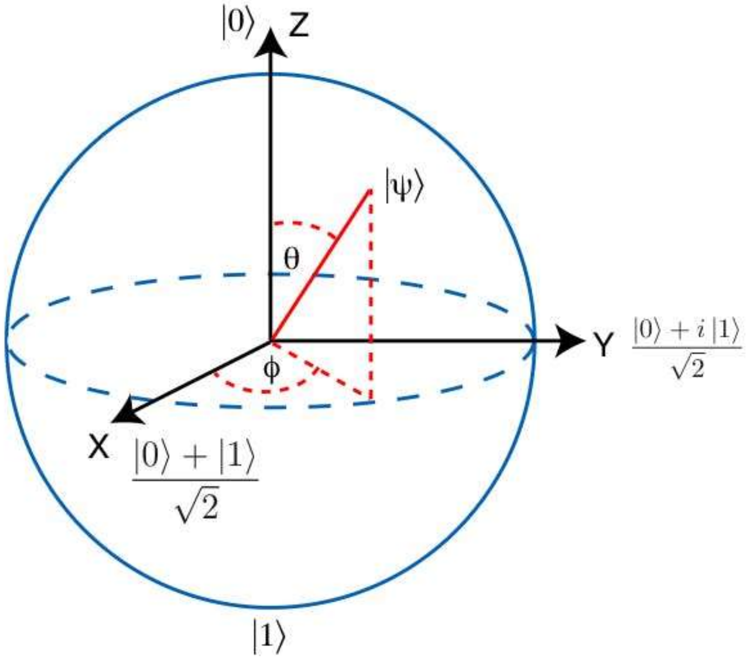
- Ideas for quantum algorithm
  - Quantum parallelism
- Deutsch-Jozsa algorithm
  - Deutsch’s problem
  - Implementation of DJ algrorithm
  - Examples
    - 1-bit
    - 2-bit (as a quiz)
    - 3-bit

Hadamard on n qubits

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$H|0\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \frac{|0\rangle + |1\rangle}{\sqrt{2}}$$

$$H|1\rangle = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \end{bmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$


$|0\rangle$
Z
$|\psi\rangle$
θ
ϕ
Y
$\frac{|0\rangle + i\,|1\rangle}{\sqrt{2}}$
X
$\frac{|0\rangle + |1\rangle}{\sqrt{2}}$
$|1\rangle$

$$|a\rangle \;-\!\!\!-\!\!\!-\boxed{H}-\!\!\!-\!\!\!-\; \frac{1}{\sqrt{2}}\sum_{b=0,1}(-1)^{a\cdot b}|b\rangle = \frac{|0\rangle + (-1)^{a}|1\rangle}{\sqrt{2}}$$

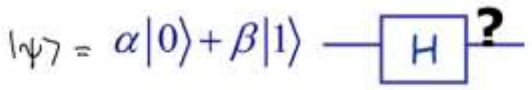
$$\begin{aligned} H|0\rangle &= \frac{1}{\sqrt{2}}\sum_{b=0,1}|b\rangle = \frac{|0\rangle + |1\rangle}{\sqrt{2}} \\ H|1\rangle &= \frac{1}{\sqrt{2}}\sum_{b=0,1}(-1)^{b}|b\rangle = \frac{|0\rangle - |1\rangle}{\sqrt{2}} \end{aligned} \quad \Leftrightarrow \quad H\begin{bmatrix}1\\0\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\1\end{bmatrix}, H\begin{bmatrix}0\\1\end{bmatrix} = \frac{1}{\sqrt{2}}\begin{bmatrix}1\\-1\end{bmatrix}$$

$$\Leftrightarrow \quad H = \frac{1}{\sqrt{2}}\begin{bmatrix}1 & 1\\1 & -1\end{bmatrix}$$

Details of matrix form

$$H|0\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}1\\0\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\0\end{pmatrix} + \frac{1}{\sqrt{2}}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)$$

$$H|1\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\1 & -1\end{pmatrix}\begin{pmatrix}0\\1\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}1\\-1\end{pmatrix} = \frac{|0\rangle - |1\rangle}{\sqrt{2}}$$



Hint: you need to work out $H|\psi\rangle = \frac{1}{\sqrt{2}}\begin{pmatrix}1&1\\1&-1\end{pmatrix}\begin{pmatrix}\alpha\\\beta\end{pmatrix} = \dots$

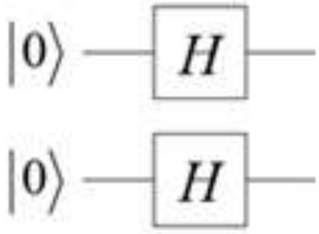
$$\frac{1}{\sqrt{2}}\begin{pmatrix}1&1\\1&-1\end{pmatrix}\begin{pmatrix}\alpha\\\beta\end{pmatrix} = \frac{1}{\sqrt{2}}\begin{pmatrix}\alpha+\beta\\\alpha-\beta\end{pmatrix} = \frac{1}{\sqrt{2}}\left(\alpha\underbrace{\begin{pmatrix}1\\0\end{pmatrix}}_{|0\rangle} + \beta\underset{|1\rangle}{\begin{pmatrix}1\\0\end{pmatrix}} + \alpha\underset{|1\rangle}{\begin{pmatrix}0\\1\end{pmatrix}} - \beta\underset{|0\rangle}{\begin{pmatrix}0\\1\end{pmatrix}}\right)$$

$$= \alpha\,\frac{|0\rangle+|1\rangle}{\sqrt{2}} + \beta\,\frac{|0\rangle-|1\rangle}{\sqrt{2}}$$

$$\alpha|0\rangle+\beta|1\rangle \xrightarrow{H} \alpha|+\rangle+\beta|-\rangle$$

2 Qubits

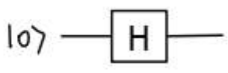
$$H = \frac{1}{\sqrt{2}}\begin{bmatrix}1&1&0&0\\1&-1&0&0\\0&0&1&1\\0&0&1&-1\end{bmatrix}$$



$$\begin{aligned}
& H|0\rangle\otimes H|0\rangle \\
= \;& \frac{1}{\sqrt{2}}\big(|0\rangle+|1\rangle\big)\otimes\frac{1}{\sqrt{2}}\big(|0\rangle+|1\rangle\big) \\
= \;& \frac{1}{2}\big(|00\rangle+|01\rangle+|10\rangle+|11\rangle\big) \\
= \;& \frac{1}{2}\big(|0\rangle+|1\rangle+|2\rangle+|3\rangle\big)=\frac{1}{2}\sum_{x=0}^{3}|x\rangle
\end{aligned}$$

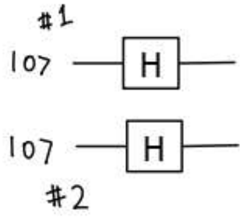
Synopsis

One qubit



$$\frac{|0\rangle+|1\rangle}{\sqrt{2}} = \sum_{k=0}^{1}\frac{1}{\sqrt{2}}|k\rangle$$

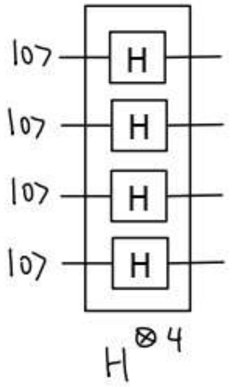
Two qubits



$$\left.\begin{matrix}\dfrac{|0\rangle+|1\rangle}{\sqrt{2}}\\[2ex] \dfrac{|0\rangle+|1\rangle}{\sqrt{2}}\end{matrix}\right\}\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right)\otimes\left(\frac{|0\rangle+|1\rangle}{\sqrt{2}}\right) =$$

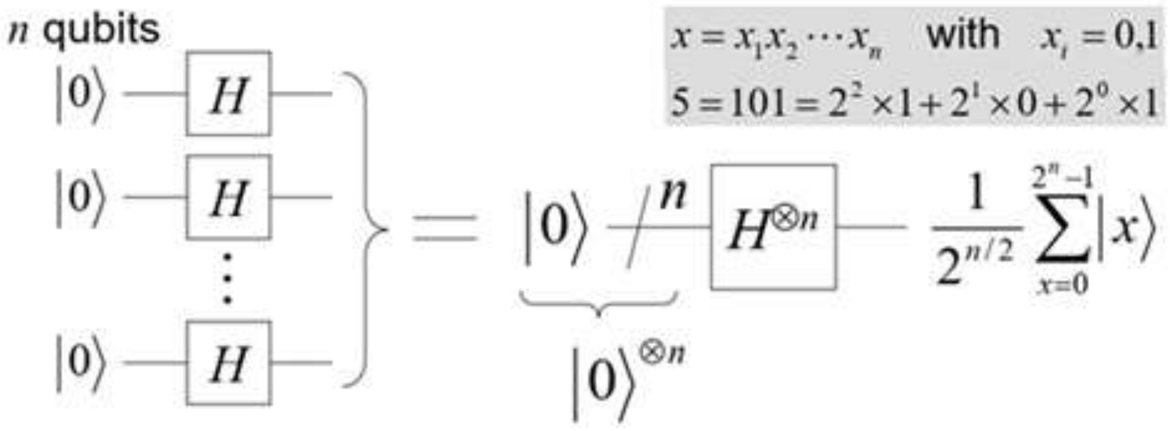
$$= \frac{1}{\sqrt{4}}\left\{|00\rangle+|01\rangle+|10\rangle+|11\rangle\right\} = \sum_{k=0}^{3}\frac{1}{\sqrt{4}}|k\rangle \quad \nwarrow 2^{n}$$

Four qubits



$$\left.\right\}\frac{1}{\sqrt{16}}\left(|0\rangle+|1\rangle\right)\left(|0\rangle+|1\rangle\right)\left(|0\rangle+|1\rangle\right)\left(|0\rangle+|1\rangle\right)$$

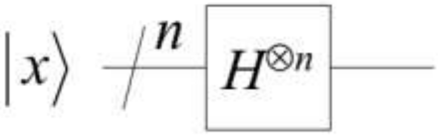
$$= \frac{1}{\sqrt{16}}\sum_{k=0}^{15}|k\rangle$$



$x = x_1x_2\cdots x_n$ with $x_i = 0{,}1$

$5 = 101 = 2^2\times1+2^1\times0+2^0\times1$

$$\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle$$



$$\frac{1}{2^{n/2}}\sum_{z}(-1)^{x\cdot z}|z\rangle$$

$$\begin{aligned}
& H^{\otimes n}|x_1\rangle|x_2\rangle\cdots|x_n\rangle \\
= \quad & \frac{1}{2^{n/2}}\left(\sum_{z_1}(-1)^{x_1\cdot z_1}|z_1\rangle\right)\cdots\left(\sum_{z_n}(-1)^{x_n\cdot z_n}|z_n\rangle\right) \\
= \quad & \frac{1}{2^{n/2}}\sum_{z_1,z_2\cdots z_n}(-1)^{x_1\cdot z_1}(-1)^{x_2\cdot z_2}\cdots(-1)^{x_n\cdot z_n}|z_1 z_2\cdots z_n\rangle \\
= \quad & \frac{1}{2^{n/2}}\sum_{z}(-1)^{x\cdot z}|z\rangle
\end{aligned}$$

$$x\cdot z \equiv x_1\cdot z_1 + x_2\cdot z_2 + \cdots + x_n\cdot z_n$$

Bitwise inner product of x and z modulo 2

Prove the following

$$H^2 = I$$

$$HZH = X$$

Use the following expressions for quantum gates

$$C_{12}|a\rangle|b\rangle = |a\rangle|b\oplus a\rangle$$

$$H|a\rangle = \frac{1}{\sqrt{2}}\sum_{b}(-1)^{a\cdot b}|b\rangle, \quad Z|a\rangle = (-1)^{a}|a\rangle$$

$$\begin{aligned}HH\left|a\right\rangle &= H\left(\frac{1}{\sqrt{2}}\sum_b(-1)^{a\cdot b}\left|b\right\rangle\right)\\ &= \frac{1}{\sqrt{2}}\sum_b(-1)^{a\cdot b}\left(\frac{1}{\sqrt{2}}\sum_c(-1)^{b\cdot c}\left|c\right\rangle\right) = \frac{1}{2}\sum_b\sum_c(-1)^{(a+c)\cdot b}\left|c\right\rangle\\ &= \frac{1}{2}\sum_b(\left|a\right\rangle+(-1)^b\left|\overline{a}\right\rangle)\\ &= \frac{1}{2}(\left|a\right\rangle+\left|\overline{a}\right\rangle+\left|a\right\rangle-\left|\overline{a}\right\rangle)=\left|a\right\rangle\end{aligned}$$

$$(a+c)\cdot b=\begin{cases}0 & (c=a)\\ b & (c=\overline{a})\end{cases}$$

Constructive and destructive interferences

$$\begin{aligned}HZH\left|a\right\rangle &= HZ\left(\frac{1}{\sqrt{2}}\sum_b(-1)^{a\cdot b}\left|b\right\rangle\right)\\ &= H\left(\frac{1}{\sqrt{2}}\sum_b(-1)^{a\cdot b+b}\left|b\right\rangle\right)\\ &= \frac{1}{2}\sum_b\sum_c(-1)^{(\overline{a}+c)\cdot b}\left|c\right\rangle\\ &= \frac{1}{2}\sum_b(\left|\overline{a}\right\rangle+(-1)^b\left|a\right\rangle)\\ &= \frac{1}{2}(\left|\overline{a}\right\rangle+\left|a\right\rangle+\left|\overline{a}\right\rangle-\left|a\right\rangle)=\left|\overline{a}\right\rangle\end{aligned}$$

$$a\cdot b+b=(a+1)\cdot b=\overline{a}\cdot b$$

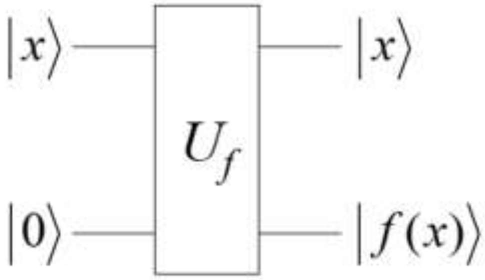
$$(\overline{a}+c)\cdot b=\begin{cases}0 & (c=\overline{a})\\ b & (c=a)\end{cases}$$

Constructive and destructive interferences

Quantum Parallelism

Suppose we are given a quantum gate U_f

$$U_f|x\rangle|y\rangle = |x\rangle|y \oplus f(x)\rangle$$



where $f(x)$ is a binary function

Remarkably, for proper inputs, we can encode all the information on $f(x)$ by applying U_f only once

$$\frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}|x\rangle|0\rangle \xrightarrow{U_f} \frac{1}{2^{n/2}}\sum_{x=0}^{2^n-1}\underbrace{|x\rangle|f(x)\rangle}_{\text{Entangled}}$$

$$\frac{1}{2}\left(|0\rangle+|1\rangle+|2\rangle+|3\rangle\right)|0\rangle$$

$$\xrightarrow{U_f} \frac{\cancel{|0\rangle|f(0)\rangle}+|1\rangle|f(1)\rangle+\cancel{|2\rangle|f(2)\rangle}+\cancel{|3\rangle|f(3)\rangle}}{2}$$

Is this useful?

The answer is ***NO***, because we must observe the state to extract information out of it, which prevents us from enjoying the full power of quantum entanglement and quantum parallelism

Quantum interference is the key

Deutsch's problem

Definition

A binary function $f(x)$ is called ***constant*** if it outputs only 0, or only 1, for all values of x

A binary function $f(x)$ is called ***balanced*** if it outputs 0 for half of all the possible x, and 1 for the other half

Constant

x	$f(x)$
0	0
1	0
2	0
3	0

Balanced

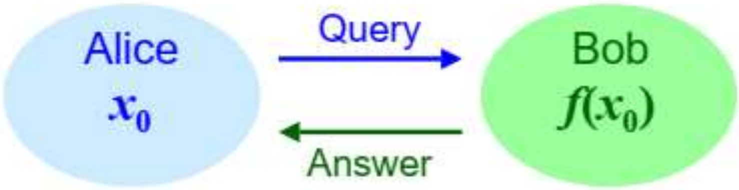
x	$f(x)$
0	0
1	0
2	1
3	1

Neither *C* or *B*

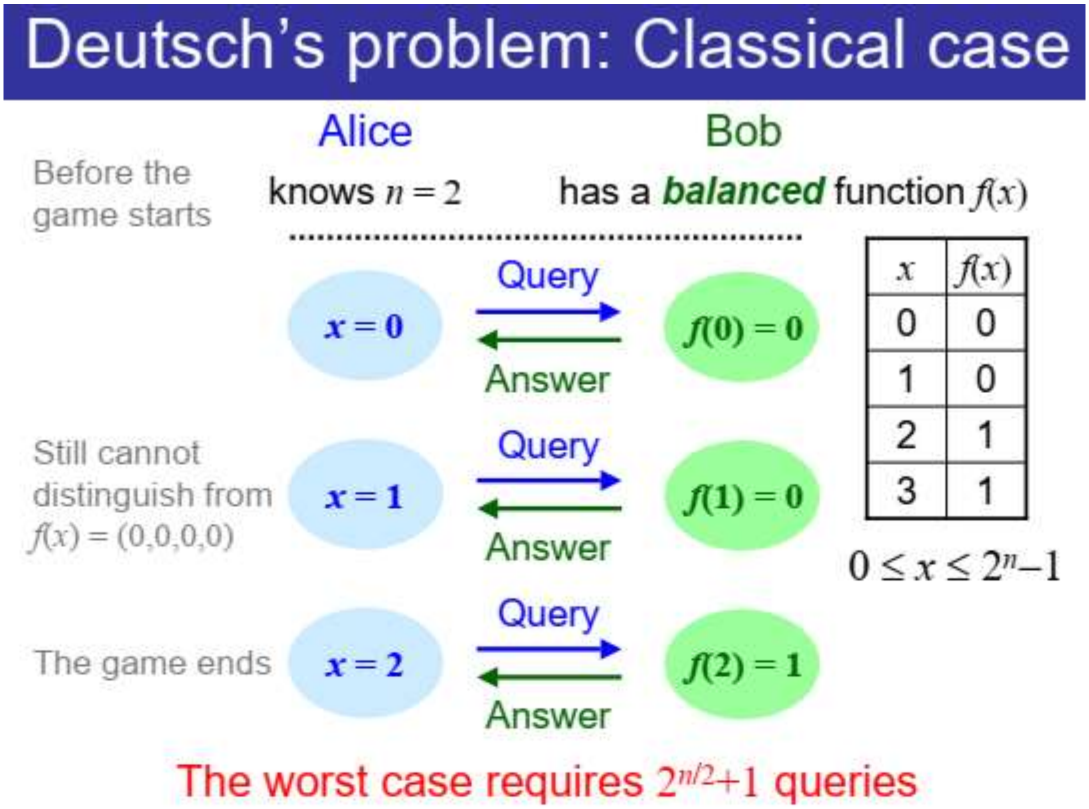
x	$f(x)$
0	0
1	0
2	0
3	1

Deutsch's problem

Constant or balanced, that is the problem



How many times does Alice have to query Bob to determine the type of his function?



Deutsch's problem: Classical case

Alice

Bob

Before the game starts

knows $n = 2$

has a ***balanced*** function $f(x)$

$x = 0$

Query

Answer

$f(0) = 0$

Still cannot distinguish from $f(x) = (0,0,0,0)$

$x = 1$

Query

Answer

$f(1) = 0$

The game ends

$x = 2$

Query

Answer

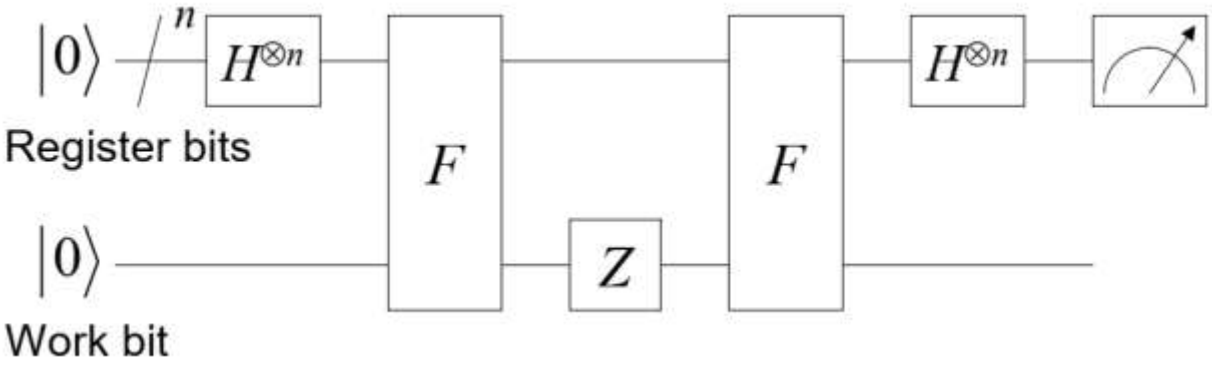
$f(2) = 1$

x	$f(x)$
0	0
1	0
2	1
3	1

$0 \le x \le 2^n - 1$

The worst case requires $2^{n/2}+1$ queries

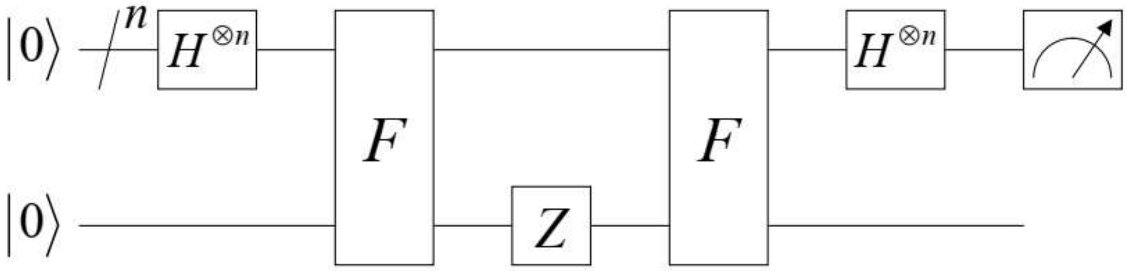
Quantum circuit for DJ



$$H^{\otimes n}|x\rangle = \frac{1}{2^{n/2}}\sum_{z}(-1)^{x\cdot z}|z\rangle \qquad F|x\rangle|w\rangle = |x\rangle|w\oplus f(x)\rangle$$

$$x\cdot z \equiv x_1\cdot z_1 + x_2\cdot z_2 + \cdots + x_n\cdot z_n \qquad Z|w\rangle = (-1)^{w}|w\rangle$$

Implementing DJ

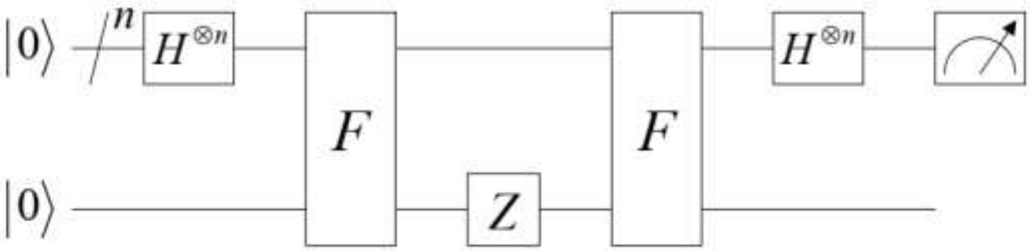


$$|0\rangle^{\otimes n}|0\rangle \xrightarrow{H^{\otimes n}} \frac{1}{2^{n/2}}\sum_{x}|x\rangle|0\rangle$$

Create a linear superposition state

$$\xrightarrow{F} \frac{1}{2^{n/2}}\sum_{x}|x\rangle|f(x)\rangle$$

Encode information on $f(x)$ into the work bit

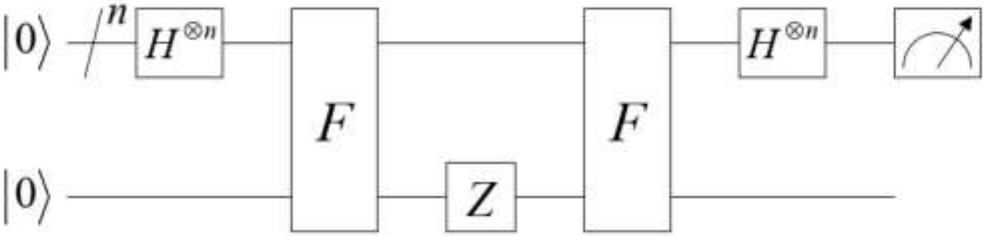


$$\frac{1}{2^{n/2}}\sum_x |x\rangle|f(x)\rangle \xrightarrow{Z} \frac{1}{2^{n/2}}\sum_x (-1)^{f(x)}|x\rangle|f(x)\rangle$$

Add nonlocal phase shifts which carry information on $f(x)$

$$\xrightarrow{F} \frac{1}{2^{n/2}}\sum_x (-1)^{f(x)}|x\rangle|0\rangle$$

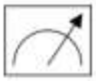
Erase information on $f(x)$ from the work bit



$$\frac{1}{2^{n/2}}\sum_x (-1)^{f(x)}|x\rangle|0\rangle \xrightarrow{H^{\otimes n}} \sum_z \boxed{\sum_x \frac{(-1)^{f(x)+x\cdot z}}{2^n}}|z\rangle|0\rangle$$

$$H^{\otimes n}|x\rangle = \frac{1}{2^{n/2}}\sum_z (-1)^{x\cdot z}|z\rangle$$

Probability amplitude for the state $|\, z\,\rangle$


$\longrightarrow$ Get $z = 0$ if and only if f is a constant function

Probability amplitude for the state $|0\rangle^{\otimes n}$

$$\sum_x \frac{(-1)^{f(x)}}{2^n} = \begin{cases} \pm 1 & \text{(constant)} \\ 0 & \text{(balanced)} \end{cases}$$

Only the constant functions bring the register back to the initial state

$n=2$, constant case

Constructive interference

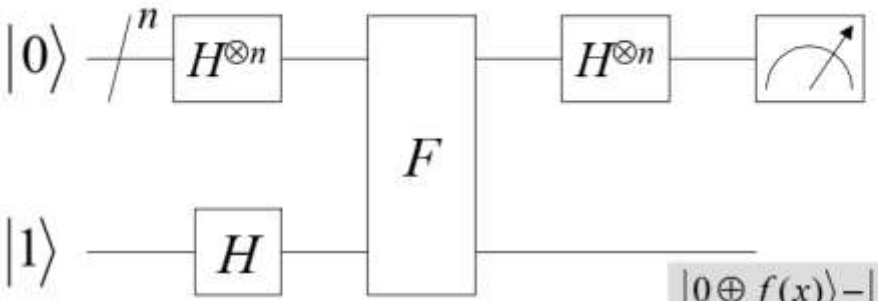
$$\sum_{x=0}^{3} \frac{(-1)^{f(x)}}{2^2} = \frac{(-1)^0+(-1)^0+(-1)^0+(-1)^0}{4} = 1$$

$n=2$, balanced case

Destructive interference

$$\sum_{x=0}^{3} \frac{(-1)^{f(x)}}{2^2} = \frac{(-1)^0+(-1)^1+(-1)^0+(-1)^1}{4} = 0$$

Revised Version



$$|0\oplus f(x)\rangle - |1\oplus f(x)\rangle$$
$$= \begin{cases} |0\rangle - |1\rangle \text{ if } f(x)=0 \\ |1\rangle - |0\rangle \text{ if } f(x)=1 \end{cases}$$
$$= (-1)^{f(x)}\left(|0\rangle - |1\rangle\right)$$

A clever choice of the work bit simplifies the circuit

$$|0\rangle^{\otimes n}|1\rangle \xrightarrow{H^{\otimes n+1}} \frac{1}{2^{n/2}}\sum_x |x\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right] \xrightarrow{F} \underline{\frac{1}{2^{n/2}}\sum_x (-1)^{f(x)}|x\rangle}\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]$$

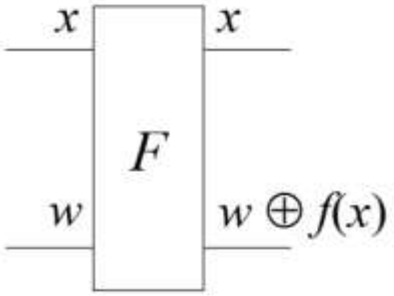
State after the 2nd F gate

$$\xrightarrow{H^{\otimes n}} \frac{1}{2^n}\sum_{x,z}(-1)^{f(x)+x\cdot z}|z\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]$$

1-bit $f(x)$

x	Constant		Balanced	
	f_{c0}	f_{c1}	f_{b0}	f_{b1}
0	0	1	0	1
1	0	1	1	0

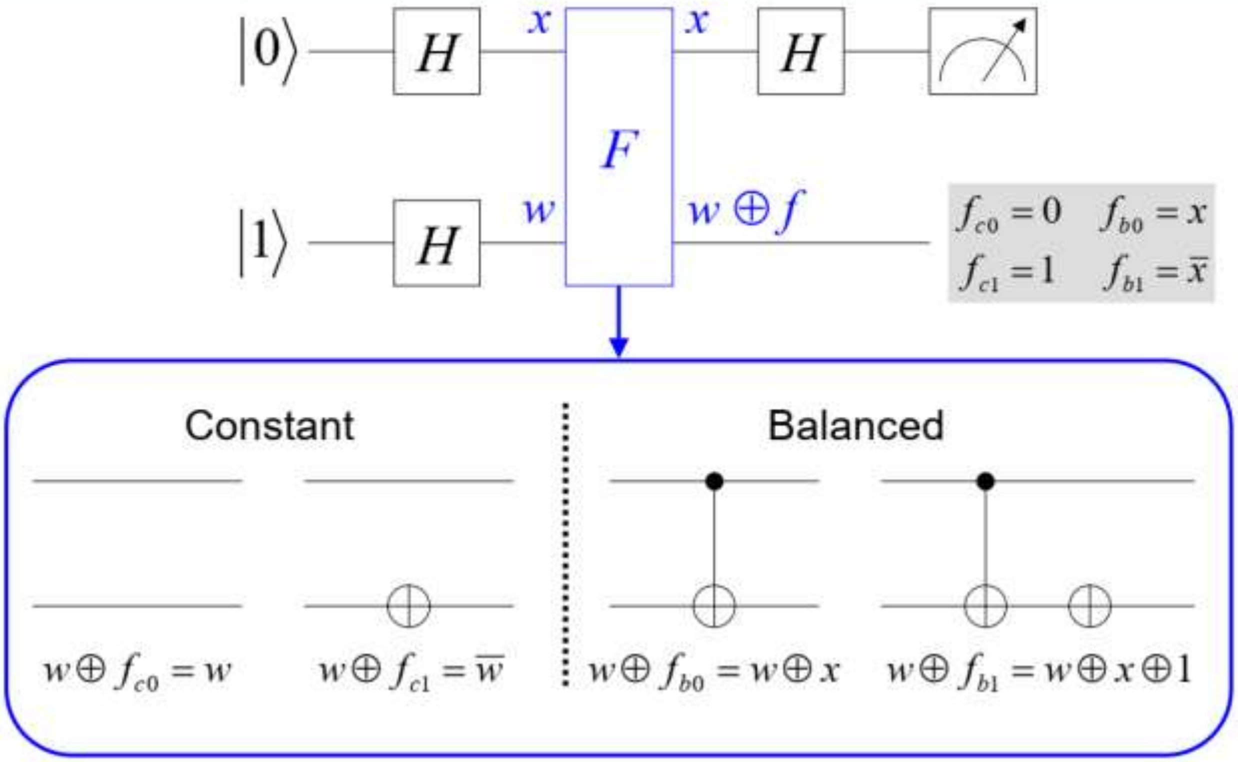
$$f_{c0}(x)=0 \quad f_{b0}(x)=x$$
$$f_{c1}(x)=1 \quad f_{b1}(x)=\overline{x}$$



$$F|x\rangle|w\rangle=|x\rangle|w\oplus f(x)\rangle$$

What is the explicit quantum circuit for the F gate?

1-bit F gate



$f_{c0} = 0$ $f_{b0} = x$

$f_{c1} = 1$ $f_{b1} = \overline{x}$

Constant

Balanced

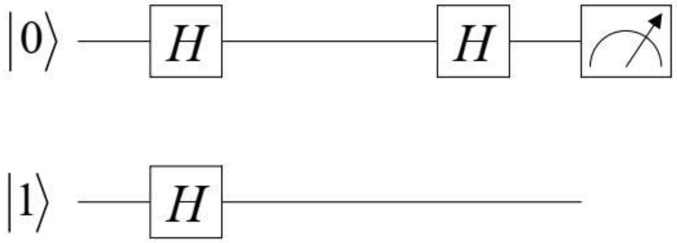
$w \oplus f_{c0} = w$

$w \oplus f_{c1} = \overline{w}$

$w \oplus f_{b0} = w \oplus x$

$w \oplus f_{b1} = w \oplus x \oplus 1$

1-bit DJ: Constant f_{c0}

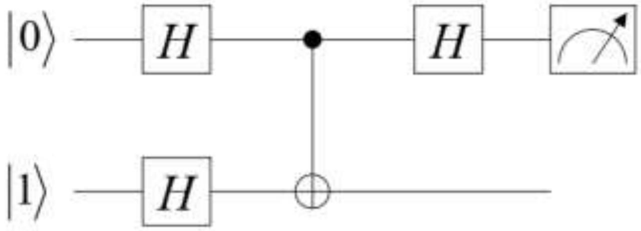


$$HH|0\rangle = \frac{1}{2}\left(|0\rangle + |1\rangle + |0\rangle - |1\rangle\right) = |0\rangle$$

Constructive interference

The initial state $|0\rangle$ “survives” due to the constructive interference, while the other state $|1\rangle$ is erased due to the destructive interference

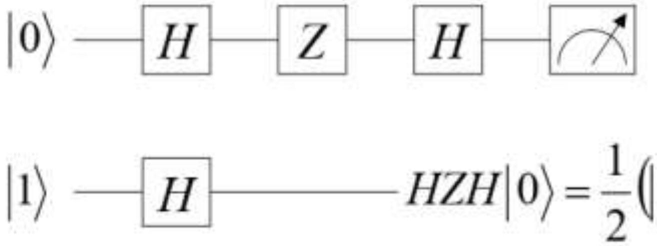
1-bit DJ: Balanced f_{b0}



$$\begin{aligned}&|0\oplus x\rangle-|1\oplus x\rangle\\&=\begin{cases}|0\rangle-|1\rangle & \text{if } x=0\\ |1\rangle-|0\rangle & \text{if } x=1\end{cases}\\&=(-1)^x\left(|0\rangle-|1\rangle\right)\end{aligned}$$

$$|0\rangle|1\rangle \xrightarrow{H^{\otimes 2}} \frac{1}{\sqrt{2}}\sum_{x=0}^{1}|x\rangle\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]\xrightarrow{C_{rw}}\frac{1}{\sqrt{2}}\sum_{x=0}^{1}\underline{(-1)^x|x\rangle}\left[\frac{|0\rangle-|1\rangle}{\sqrt{2}}\right]$$

Z gate on the register



Destructive interference

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

2-bit $f(x)$

x	ab	Constant		Balanced $({}_4C_2 = 6)$					
		f_{c0}	f_{c1}	f_{b0}	f_{b1}	f_{b2}	f_{b3}	f_{b4}	f_{b5}
0	00	0	1	0	0	0	1	1	1
1	01	0	1	0	1	1	1	0	0
2	10	0	1	1	0	1	0	1	0
3	11	0	1	1	1	0	0	0	1

$$\begin{array}{lll}
f_{c0}(x) = 0 & f_{b0}(x) = a & f_{b3}(x) = \overline{a} \\
f_{c1}(x) = 1 & f_{b1}(x) = b & f_{b4}(x) = \overline{b} \\
 & f_{b2}(x) = a \oplus b & f_{b5}(x) = \overline{a \oplus b}
\end{array}$$

2-bit F gates can be constructed from only CNOT and NOT

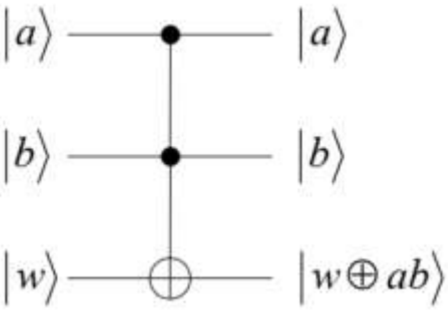
3-bit balanced $f(x)$

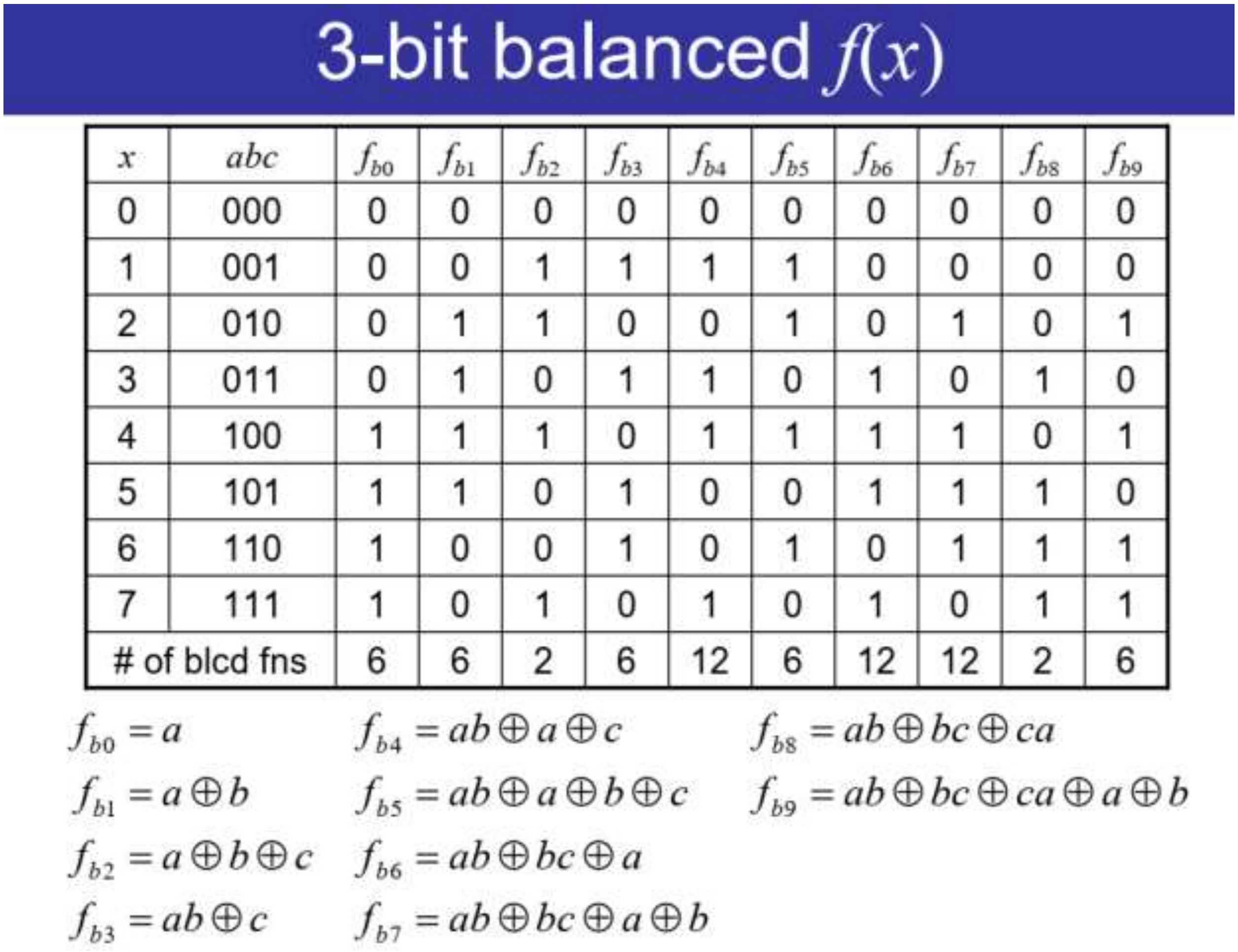
$$f_{b0} = a$$
$$f_{b1} = a \oplus b$$
$$f_{b2} = a \oplus b \oplus c$$
$$f_{b3} = ab \oplus c$$
$$f_{b4} = ab \oplus a \oplus c$$
$$f_{b5} = ab \oplus a \oplus b \oplus c$$
$$f_{b6} = ab \oplus bc \oplus a$$
$$f_{b7} = ab \oplus bc \oplus a \oplus b$$
$$f_{b8} = ab \oplus bc \oplus ca$$
$$f_{b9} = ab \oplus bc \oplus ca \oplus a \oplus b$$

Number of balanced functions

$${}_8C_4 = 70$$

3-bit F gates require not only CNOT but Toffoli





3-bit balanced $f(x)$

x	abc	f_{b0}	f_{b1}	f_{b2}	f_{b3}	f_{b4}	f_{b5}	f_{b6}	f_{b7}	f_{b8}	f_{b9}
0	000	0	0	0	0	0	0	0	0	0	0
1	001	0	0	1	1	1	1	0	0	0	0
2	010	0	1	1	0	0	1	0	1	0	1
3	011	0	1	0	1	1	0	1	0	1	0
4	100	1	1	1	0	1	1	1	1	0	1
5	101	1	1	0	1	0	0	1	1	1	0
6	110	1	0	0	1	0	1	0	1	1	1
7	111	1	0	1	0	1	0	1	0	1	1
# of blcd fns		6	6	2	6	12	6	12	12	2	6

$f_{b0} = a$

$f_{b1} = a \oplus b$

$f_{b2} = a \oplus b \oplus c$

$f_{b3} = ab \oplus c$

$f_{b4} = ab \oplus a \oplus c$

$f_{b5} = ab \oplus a \oplus b \oplus c$

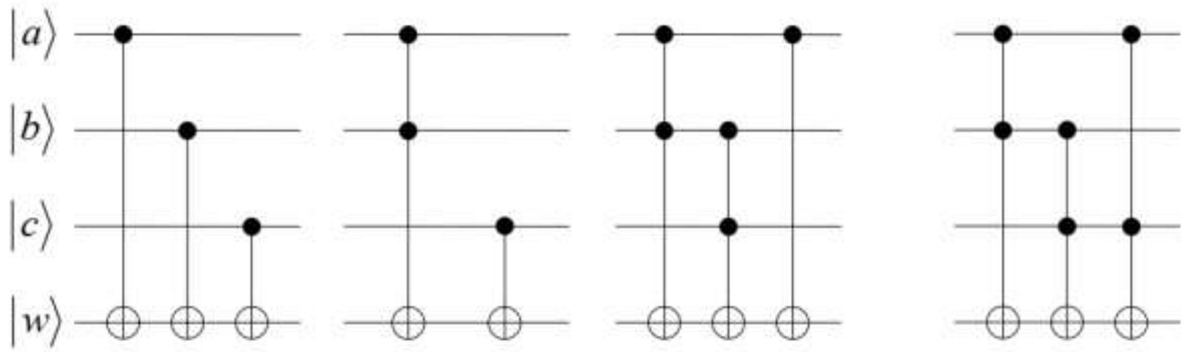
$f_{b6} = ab \oplus bc \oplus a$

$f_{b7} = ab \oplus bc \oplus a \oplus b$

$f_{b8} = ab \oplus bc \oplus ca$

$f_{b9} = ab \oplus bc \oplus ca \oplus a \oplus b$

3-bit DJ: Balanced

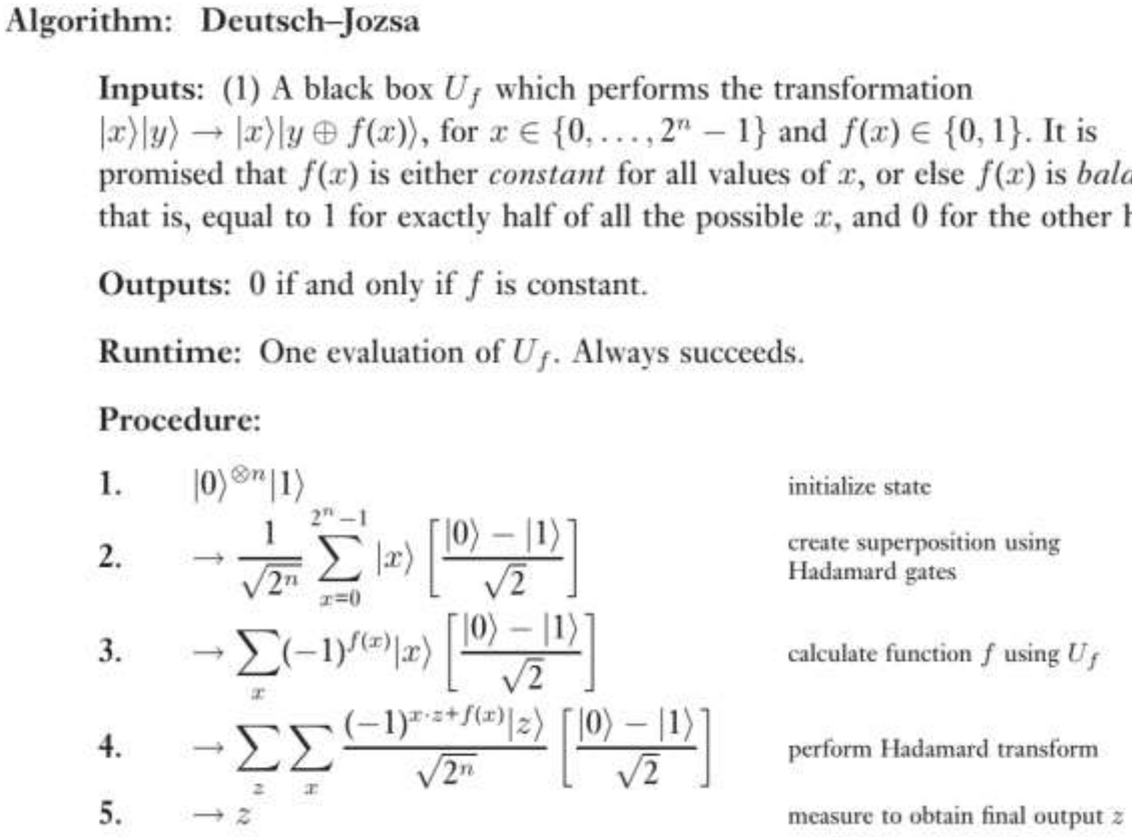


$w \oplus f_{b2}$ $= w \oplus a \oplus b \oplus c$

$w \oplus f_{b3}$ $= w \oplus ab \oplus c$

$w \oplus f_{b6}$ $= w \oplus ab \oplus bc \oplus a$

$w \oplus f_{b8}$ $= w \oplus ab \oplus bc \oplus ca$



Algorithm: Deutsch–Jozsa

Inputs: (1) A black box U_f which performs the transformation $|x\rangle|y\rangle \rightarrow |x\rangle|y \oplus f(x)\rangle$, for $x \in \{0, \ldots, 2^n - 1\}$ and $f(x) \in \{0, 1\}$. It is promised that $f(x)$ is either *constant* for all values of x, or else $f(x)$ is *balanced*, that is, equal to 1 for exactly half of all the possible x, and 0 for the other half.

Outputs: 0 if and only if f is constant.

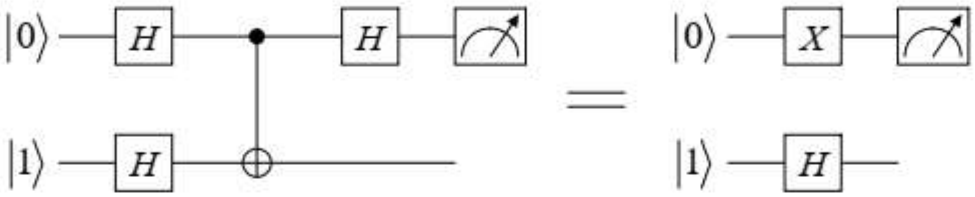
Runtime: One evaluation of U_f. Always succeeds.

Procedure:

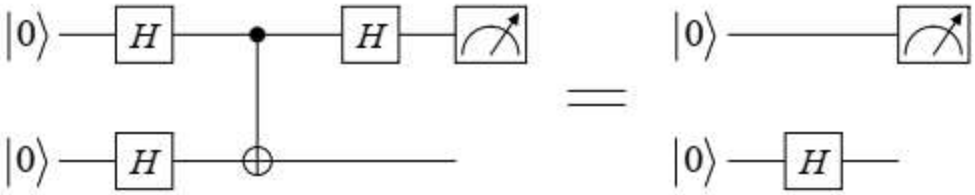
1. $|0\rangle^{\otimes n}|1\rangle$ initialize state
2. $\rightarrow \frac{1}{\sqrt{2^n}} \sum_{x=0}^{2^n-1} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$ create superposition using Hadamard gates
3. $\rightarrow \sum_x (-1)^{f(x)} |x\rangle \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$ calculate function f using U_f
4. $\rightarrow \sum_z \sum_x \frac{(-1)^{x \cdot z + f(x)} |z\rangle}{\sqrt{2^n}} \left[\frac{|0\rangle - |1\rangle}{\sqrt{2}}\right]$ perform Hadamard transform
5. $\rightarrow z$ measure to obtain final output z

Quiz 1

Prove the following circuit identity by converting the circuit sequentially



Also show that X in the upper line vanish if the initial state of the second qubit is $|0\rangle$



Quiz 2

Construct all the 2-bit F gates based on the list below

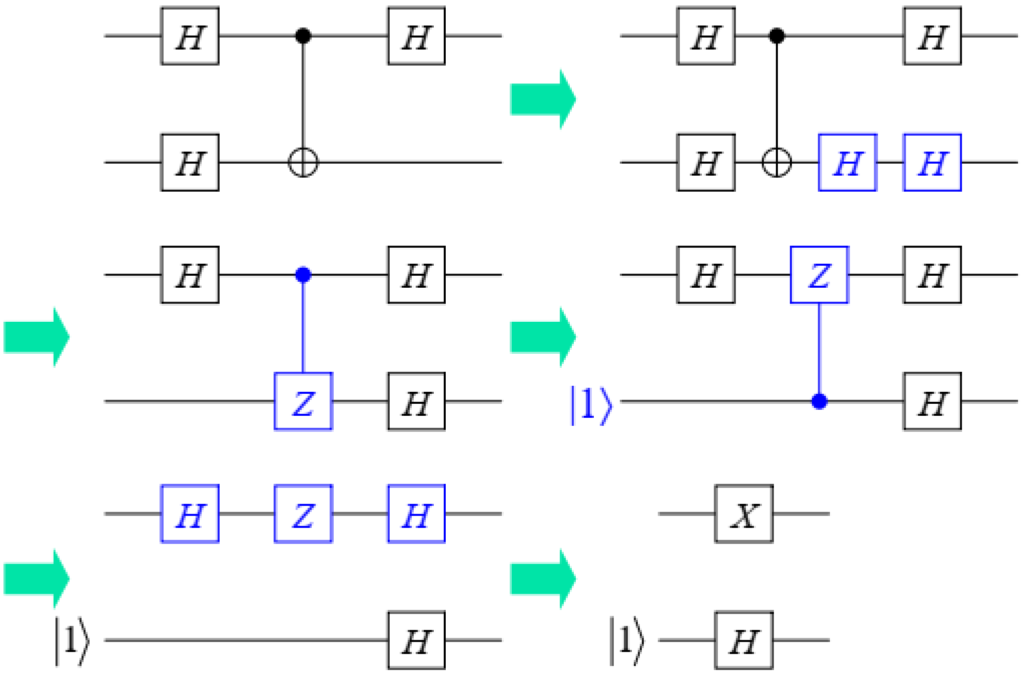
x	ab	Constant		Balanced					
		f_{c0}	f_{c1}	f_{b0}	f_{b1}	f_{b2}	f_{b3}	f_{b4}	f_{b5}
0	00	0	1	0	0	0	1	1	1
1	01	0	1	0	1	1	1	0	0
2	10	0	1	1	0	1	0	1	0
3	11	0	1	1	1	0	0	0	1

$$f_{c0}(x)=0 \quad f_{b0}(x)=a \quad f_{b3}(x)=\overline{a}$$

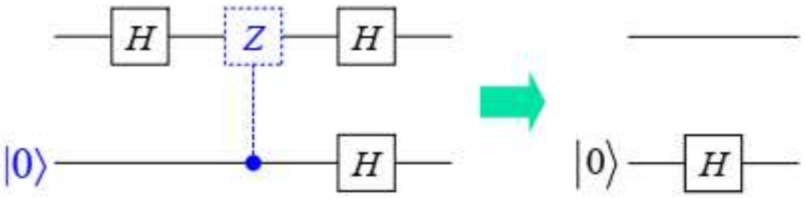
$$f_{c1}(x)=1 \quad f_{b1}(x)=b \quad f_{b4}(x)=\overline{b}$$

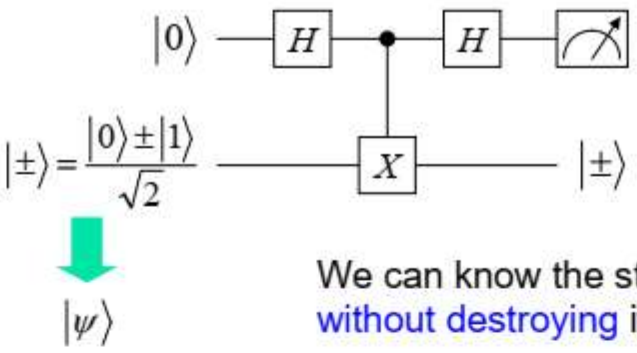
$$f_{b2}(x)=a\oplus b \quad f_{b5}(x)=\overline{a\oplus b}$$

Answer



Answer





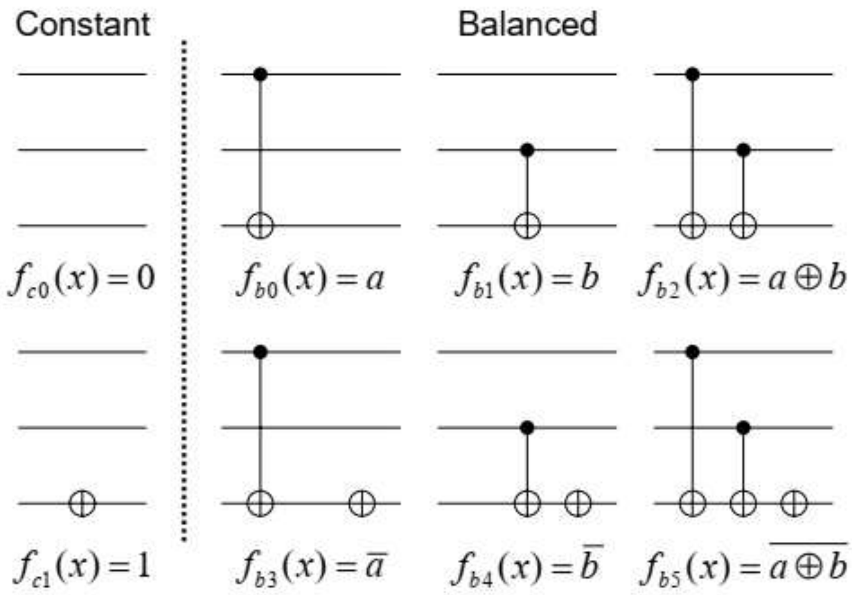
0 if the 2nd qubit is $|+\rangle$

1 if the 2nd qubit is $|-\rangle$

$$X|\pm\rangle = \pm|\pm\rangle$$

We can know the state of the 2nd qubit without destroying it (Measurement of X)

Answer



Bernstein-Vazirani Algorithm

Now consider the following function.

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

$$\mathbf{x} \to (\mathbf{a}\cdot\mathbf{x} + b) \pmod 2$$

$$(\mathbf{a} \in \{0,1\}^n, b \in \{0,1\})$$

The promise is that f conforms to the above constraints.
Our task is to determine the values of a and b. (We note that the Bernstein-Vazirani promise is a restriction on Deutsch-Josza, but the question is harder to solve.)

With a classical deterministic algorithm, n + 1 queries are required;
1 to determine b (query with x = 0), and n to determine a.

$f(000:::0_n) = b$

$f(100:::0_n) = a1$

$f(010:::0_n) = a2$

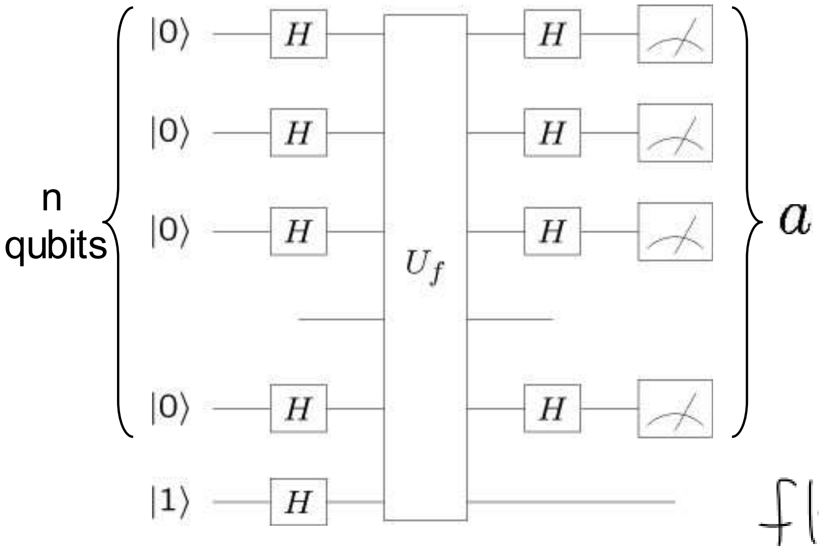
. .

$f(000:::1) = an$

(We also note that is, classically, the optimal algorithm, as n + 1 bits of information are required.) A classical randomized algorithm will do no better, also requiring n + 1 queries.

Exercise . *Show that any classical randomized algorithm that solves the Bernstein-Vazirani problem with reasonable error makes at least* $n + 1$ *queries.*

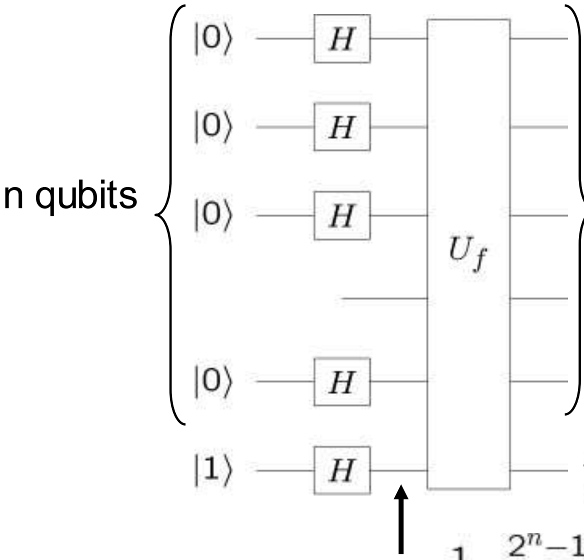
Quantum Bernstein-Vazirani



$f(x) = a \cdot x \oplus b$

Quantum interference : H -Uf-H-Measure

Quantum Bernstein-Vazirani



$$\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{f(y)}|y\rangle$$

$$=\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{a\cdot y+b}|y\rangle$$

$$\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$

$$H^{\otimes n}|0\rangle\otimes H|1\rangle=\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}|y\rangle\frac{1}{\sqrt{2}}(|0\rangle-|1\rangle)$$

Hadamard It!

$$H^{\otimes n}\left[\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{a\cdot y+b}|y\rangle\right]=\frac{1}{2^n}\sum_{x,y=0}^{2^n-1}(-1)^{a\cdot y+b}(-1)^{x\cdot y}|x\rangle$$

$$=\frac{(-1)^b}{2^n}\sum_{x=0}^{2^n-1}\left(\sum_{y=0}^{2^n-1}(-1)^{a\cdot y+x\cdot y}\right)|x\rangle$$

$$\sum_{y=0}^{2^n-1}(-1)^{a\cdot y+x\cdot y}=\sum_{y_1,\ldots,y_n=0}^{1}(-1)^{a_1y_1+\cdots+a_ny_n+x_1y_1+\cdots+x_ny_n}$$

$$=\sum_{y_1=0,1}\cdots\sum_{y_n=0,1}(-1)^{a_1y_1+\cdots+a_ny_n+x_1y_1+\cdots+x_ny_n}$$

$$=((-1)^0+(-1)^{a_1+x_1})\sum_{y_2=0,1}\cdots\sum_{y_n=0,1}(-1)^{a_2y_2+\cdots+a_ny_n+x_2y_2+\cdots+x_ny_n}$$

$$=2^n\delta_{a_1,x_1}\delta_{a_2,x_2}\cdots\delta_{a_n,x_n}=2^n\delta_{a,x}$$

$$H^{\otimes n}\left[\frac{1}{\sqrt{2^n}}\sum_{y=0}^{2^n-1}(-1)^{a\cdot y+b}|y\rangle\right]=(-1)^b\sum_{x=0}^{2^n}\delta_{a,x}|x\rangle=(-1)^b|a\rangle$$

$(a+x).y$ $a=x$ $a+x=0$ $a\neq x$ $a+x=1$

Bernstein-Vazirani

1993: Bernstein-Vazirani Algorithm
(non-recursive)

Exact classical q. complexity: n

Bounded error classical q. complexity: $\Omega(n)$

Exact quantum q. complexity: 1

Bernstein-Vazirani Algorithm

(the non-recursive) Bernstein-Vazirani Algorithm [BV97], no exponential speed up compared to the classical algorithm but a polynomial one.

Given: A function with n bit strings as input and one bit as output (i.e. Given an oracle access to)

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

and a **Promise:** That the function is of the form

$$f(x) = s \cdot x \; (\textstyle\sum_{i=1}^{n} s_i \cdot x_i) \text{ in } F_2^n$$

Problem: Find the n bit string S ,where s is a secret string that the algorithm is trying to learn.

Classically brute force and try n inputs, i.e.

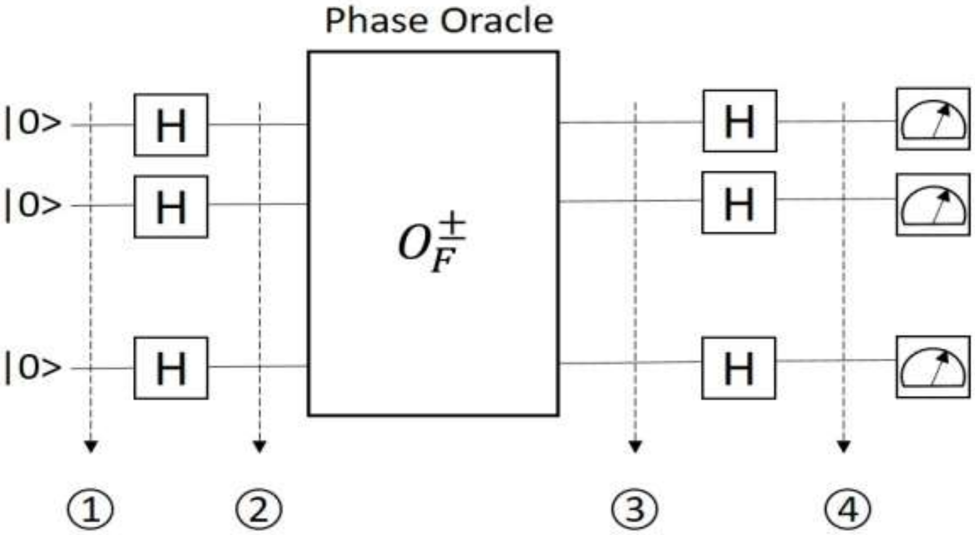
f(100:::0_n) = *s*1

f(010:::0_n) = *s*2

. .

f(000:::1) = *sn*

one query in quantum computation:



The states at different level are

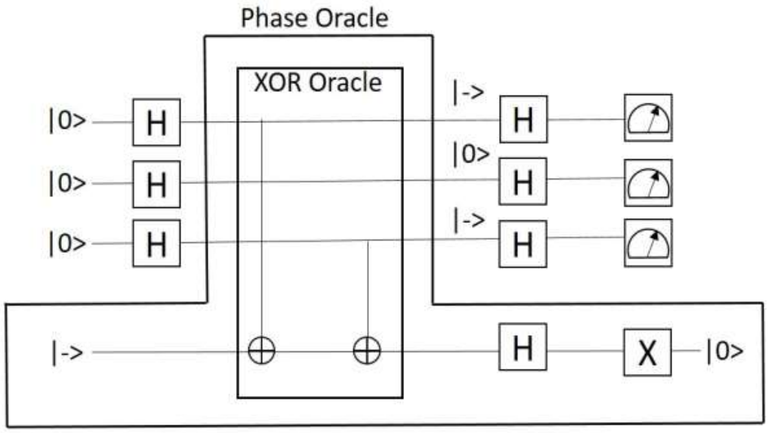
$$\textcircled{1} \rightarrow |0\rangle^{\otimes n}$$
$$\textcircled{2} \rightarrow |+\rangle^{\otimes n} = \frac{1}{\sqrt{2}} \sum_{x\in\{0,1\}^n} |x\rangle$$
$$\textcircled{3} \rightarrow \frac{1}{\sqrt{2^n}} \sum_x (-1)^{f(x)} |x\rangle = \frac{1}{\sqrt{2^n}} \sum_x (-1)^{s\cdot x} |x\rangle$$
$$= \tfrac{1}{2}(|0\rangle + (-1)^{s_1} |1\rangle) \otimes \tfrac{1}{2}(|0\rangle + (-1)^{s_2} |1\rangle) \otimes ..$$

So depending on s_i being 0 or 1, we will get $|+\rangle$ or $|-\rangle$.

$\textcircled{4} \rightarrow 0$ if $s_i = 0$ and 1 if $s_i = 1$.

Here we assume that the such an oracle exist without worrying about the practical possibilities.

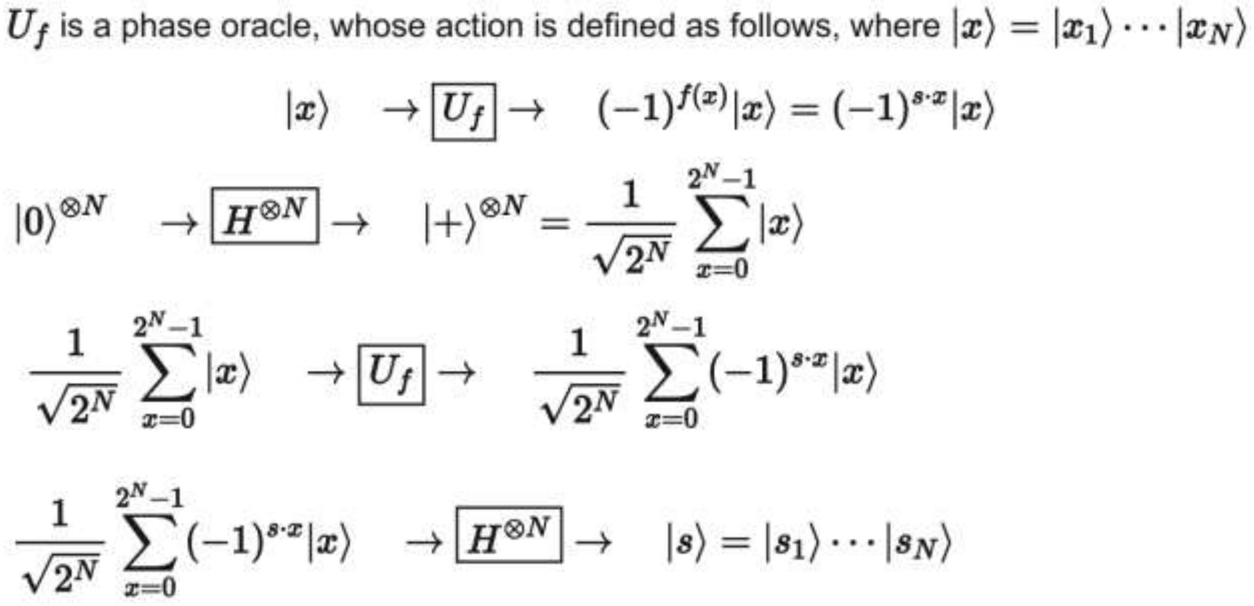
Example: with s=101 and with phase oracle built using an XOR oracle.



Here Bernstein-Vazirani algorithm only has polynomial speed up. But a modified version of this algorithm, Recursive Bernstein-Vazirani Algorithm has exponential speed up.

Usually algorithms assume that the oracles can be implemented, but in many cases implementing oracles can be difficult.

So we assume oracle is a given black box.



U_f is a phase oracle, whose action is defined as follows, where $|x\rangle = |x_1\rangle \cdots |x_N\rangle$

$$|x\rangle \quad \rightarrow \boxed{U_f} \rightarrow \quad (-1)^{f(x)}|x\rangle = (-1)^{s\cdot x}|x\rangle$$

$$|0\rangle^{\otimes N} \quad \rightarrow \boxed{H^{\otimes N}} \rightarrow \quad |+\rangle^{\otimes N} = \frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} |x\rangle$$

$$\frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} |x\rangle \quad \rightarrow \boxed{U_f} \rightarrow \quad \frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} (-1)^{s\cdot x}|x\rangle$$

$$\frac{1}{\sqrt{2^N}} \sum_{x=0}^{2^N-1} (-1)^{s\cdot x}|x\rangle \quad \rightarrow \boxed{H^{\otimes N}} \rightarrow \quad |s\rangle = |s_1\rangle \cdots |s_N\rangle$$

Big-O only stresses on bounding above. It does not bother about how far above. So an algorithm has infinite functions as its Big-O complexity. The most informative of them is the one that is immediately above, otherwise every algo is O(infinity).

Now with the definition being clear, we can easily say that Merge Sort is O(nLog(n)) as well as O(n2)

Similar to big O notation, big Omega(Ω) function is used in **computer science to describe the performance or complexity of an algorithm**. If a running time is Ω(f(n)), then for large enough n, the running time is at least k·f(n) for some constant k

By Pictures

s Big-Oh (most commonly used)

□ bounded above

s Big-Omega

□ bounded below

s Big-Theta

□ exactly

s Small-o

□ not as expensive as ...