ABSTRACT

This paper compares Quantum Algorithms to their classical counterparts, axplains the advantge of using them, and gives detailed information about building those Quantum Algorithms. In this paper two of the most important Quantum Algorithms, Deutch-Jozsa and Grover's Algorithm are reviewed. The methodology was to break two Quantum Algoriths into pieces and explain the function, and the reason behind each part of the Algorithms. The results for the comparing two Quantum Algorithms to their best Classical counterparts showed that Quantum Algirthms were significantly faster in solving complex computational problems.

Introduction to Deusch-Jozsa Algorithm

The Deutsch-Jozsa algorithm was first introduced by David Deutsch and Richard Jozsa in 1992 (David Deutsch and Richard Jozsa 1992, 553–558). It is the first example of a quantum algorithm that outperforms the best classical counterpart available. Even though it is a toy problem* Deutsch-Jozsa Algorithm showed that there can be advantages using a Quantum Computer than a Classical Computer for solving spesific types of problems.

*Toy problem: a made up problem to a trait of a spesific problem

Deutsch-Jozsa Algorithm & Oracles

We assume we have access to an oracle, a physicial device that we cannot look inside, to which we can pass queries and which returns answers

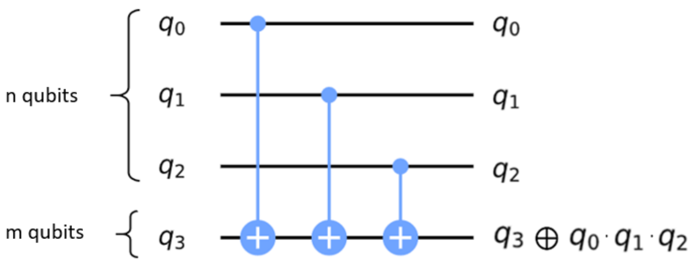
Goal: determine some property of the oracle using the minimum number of queries

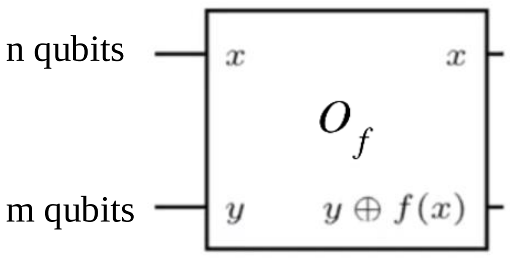
-On a Classical Computer an Oracle is given by a function where n is the length of the input string and m is the length of output string:

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

On a Quantum Computer the Oracle must be reversible meaning we should be able to trace back our results to the inputs we give. Here we have an oracle with n input qubits and m output qubits. The input of n qubits will not change any state but we will still keep them for our oracle to be reversible. m qubits will be our output qubits, the oracle implemented on x will be $f(x)$, and since they will be stored in m qubits our output will be $y \oplus f(x)$.

For example:
we can create a random oracle like this and we can observe how our oracle behaves. The only difference with this figure and the one below is that we can see what is inside this spesific oracle here.

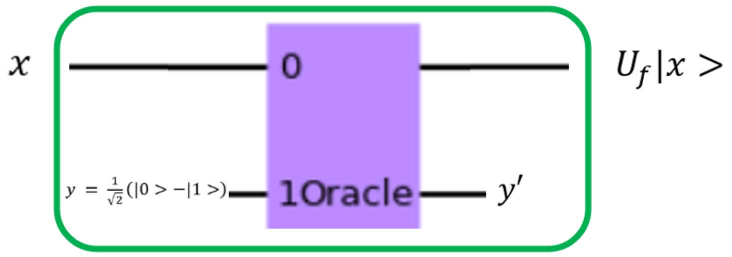




O_f: Bit Oracle, can be seen as a unitary which performs the map:

$$O_f \,|x> \,|y> \;=\; |x> \,|\, y \oplus f(x) >$$

For $f : \{0,1\}^n \rightarrow \{0,1\}$ we can construct U_f :



$$O_f\,|x>\,|y> \;=\; \frac{1}{\sqrt{2}}\left(O_f|x>|0> - O_f|x>|1>\right) = \frac{1}{\sqrt{2}}\left(|x>|0\oplus f(x)> - |x>|1\oplus f(x)>\right)$$

$$= \begin{cases} \dfrac{1}{\sqrt{2}}\,|x>(\,|0> - |1>) = |x>|y>,\; if\;\; f(x) = 0 \\ \dfrac{1}{\sqrt{2}}\,|x>(\,|1> - |0>) = -\,|x>|y>,\; if\;\; f(x) = 1 \end{cases}$$

$$= (-1)^{f(x)}\,|x>|y>$$

Independent of $|y> \rightarrow U_f$: *Phase Oracle* which performs the map:

$$U_f|x> \;=\; (-1)^{f(x)}\,|x>$$

Hadamard on “n” qubits

Hadamard gate is used to create superposition in qubits. When we apply a hadamard gate to a qubit we create an equal superposition, causing our state to collapse to either 1 or 0 with 50 percent probability to each.

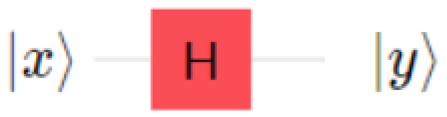
Remember that when we apply the Hadamard gate to state 0 we get a plus state, when we apply Hadamard gate to 0 we get a minus state:

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

$H|0\rangle = |+\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle+|1\rangle)$ and $H|1\rangle = |-\rangle = \frac{1}{\sqrt{2}}\,(|0\rangle- |1\rangle)$

When Hadamard gate is applied to one qubit we get a plus or minus state depending on the input value:

for $x \in \{0, 1\}$:



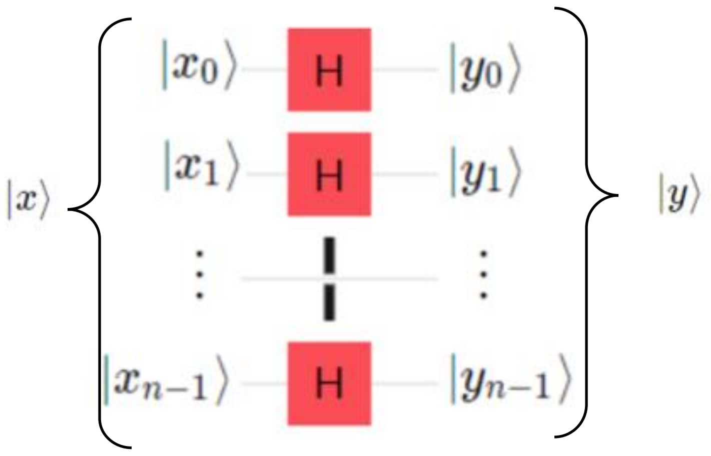
$$|y\rangle = \frac{1}{\sqrt{2}}[|0> + (-1)^{x} \cdot |1>]$$

$$= \frac{1}{\sqrt{2}}[(-1)^{0\cdot x}|0> + (-1)^{1\cdot x}|1>]$$

$$\frac{1}{\sqrt{2}} \sum_{k\in\{0,1\}} (-1)^{k\cdot x} |k>$$

When Hadamard gate is applied to multiple qubits we have a superposition of all qubits

for $x \in \{0, 1\}^n$:



inner product

$$|y> = H^{\otimes n}|x> = \frac{1}{\sqrt{2^n}} \sum_{\substack{k\in\{0,1\}\\ k\in\{0,1\}^n}} (-1)^{k\cdot x} |k>$$

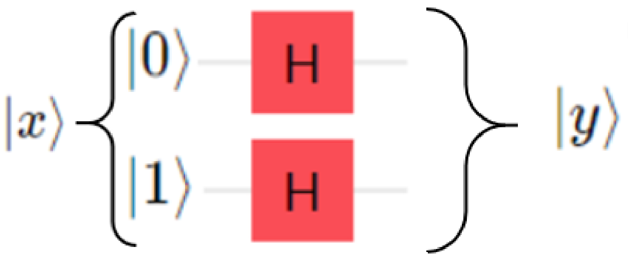
All $|y_i>$ are either $|+>$ or $|->$

$|y>$ must be a superposition of all possible 2^n bit strings

Example:

$$|y> \; = \; H^{\otimes n}|x> \; = \; \frac{1}{\sqrt{2^n}} \sum_{\substack{k \in \{0,1\} \\ k \in \{0,1\}^n}} (-1)^{\overset{\substack{\text{inner product} \\ \downarrow}}{k \cdot x}} |k>$$

$|x> \; = \; |01>$



$$|y> \; = \; \frac{1}{\sqrt{2^2}} \left[(-1)^{(0\ 0)\binom{0}{1}} |00> \; + (-1)^{(0\ 1)\binom{0}{1}} |01> \; + \; (-1)^{(1\ 0)\binom{0}{1}} |10> \; + \; (-1)^{(1\ 1)\binom{0}{1}} |11> \right]$$

$$= \frac{1}{2} (|00> - |01> + |10> - |11>)$$

Deutsch-Jozsa Algorithm

We are given a function $f : \{0, 1\}^n \rightarrow \{0, 1\}$

Realised by an oracle. The oracle is either constant, all inputs map to the same output, or balanced, half of the inputs map to “1” and the other half to “0”.

Goal: determine whetherour function f is constant or balanced

Example of a Balanced Oracle:

Inputs	**Outputs**
000	0
001	0
010	0
100	0
101	1
110	1
011	1
111	1

Classical solution: We need to ask the oracle at least twice but if we get the same output we need to ask again and again

at most (in the worst case) $N = 2^n$, $\frac{N}{2} + 1 = 2^{n-1} + 1$ queries.
(n: # input bits)

Example:
We have a ball dying machine (oracle) and it has 3 buttons on it. One buttons causes all balls to be painted white (constant function), other button dyes half of the balls to white and other half to black (balanced function), and the other button dyes all balls to black (constant function). We closed our eyes and randomly pushed a button. We want to know which button we pushed causing all the balls to be painted in the same color or half of them are painted white and the other half is painted black.

Classical Solution:
Lets say black equals the value 0, and white is 1. We want to know which button we pushed. If we pushed the button in the middle the outputs of our machine will give half of the time 0, and half of the time 1. Lets have 6 balls to be painted. The worst case to check our result would be if first half of the balls are the same color because we need to check all first half plus one to understand if our function is constant or balanced.

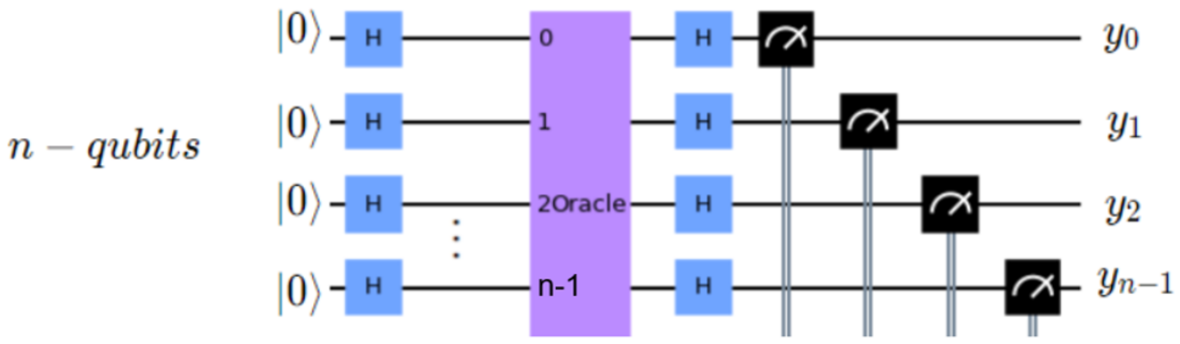
000111

We are checking first half of the outputs plus one more output. So we need to check 4 outputs before we can decide which button we pushed (deciding whether the oracle is balanced or constant). The solution to this can be represented like:

for $N = 2^n$, (n: # input bits)

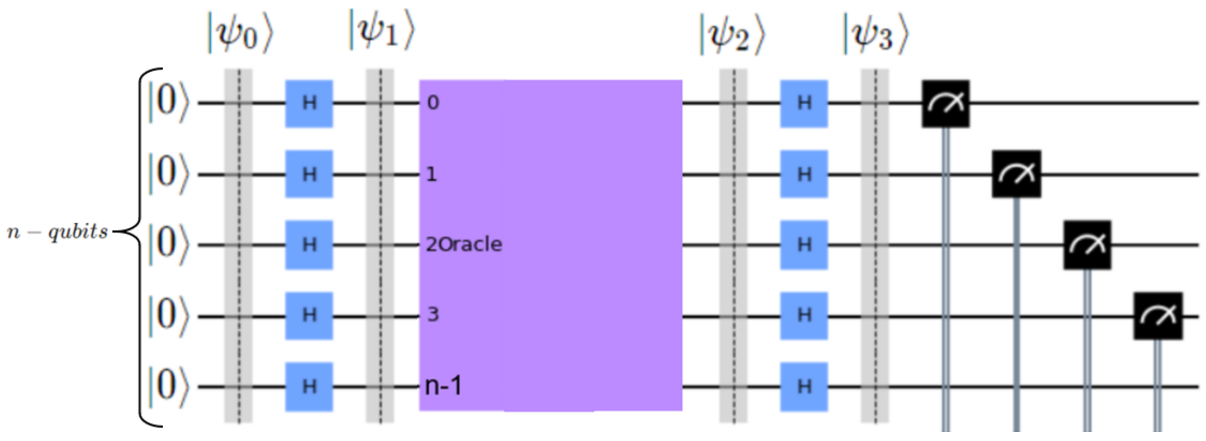
$$\frac{N}{2} + 1 = 2^{n-1} + 1$$

Quantum Solution: only one query is needed



Claim: If the outcome y equals the bit string with the length of n: 000...00
Then f is constant, otherwise it is balanced.

Proof of Deutsch-Jozsa Algorithm



Now we will check each state we labeled in the quantum circuit above

First check the state $|\psi_0>$:

$$|\psi_0> = |00\ldots0> = |0>^{\otimes n}$$

Now we apply our Hadamard Gates to each qubit and get an equal superposition of all qubits.

$$|\psi_1> \; = H^{\otimes n} \, |\psi_0> \; = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{\,x \cdot \psi_0} \; |x> \; = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x>$$

Now we apply our oracle to our equally superpositioned qubits

$$|\psi_2> \; = U_f \cdot |\psi_1> \; = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} U_f \; |x> \; = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \cdot |x>$$

Now we again apply Hadamrd Gates to our qubits to measure qubits without superposition

$$|\psi_3> \; = H^{\otimes n} \cdot \; |\psi_2> \; = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \cdot \; H^{\otimes n} \, |x>$$

$$= \frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)} \cdot \sum_{k \in \{0,1\}^n} (-1)^{\,k \cdot x} \cdot \; |k>$$

$$= \sum_{k \in \{0,1\}^n} \left[\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x) + k \cdot x} \right] \cdot |k> \; = \sum_{k \in \{0,1\}^n} c_k \, |k>$$

We have our represenatiton. Now we will look at the last part where we check the probability to measure the bit string zero

Probability to measure the zero string |000...0> :

$$\mathbb{P}\left[y = 00\ldots0\right] = |<00\ldots0|\psi_3>|^2 = |\sum_{k \in \{0,1\}^n} c_k \underbrace{<00\ldots0|k>}_{=\begin{cases} 1 \,,\; k = 00\ldots0 \\ 0 \,,\; k \neq 00\ldots0 \;\; (orthogonal) \end{cases}} |^2 = |\, c_{00\ldots0} \,|^2$$

If our k value equals to 00…0 bit string the inner product will have a value of 1 as a result however if the bit string k equals anything other than bit string 00…0 the value of that inner product will equal to 0 as a result.

$$|\underbrace{\frac{1}{2^n} \sum_{x \in \{0,1\}^n} (-1)^{f(x)}}_{=\begin{cases} +2^n \;,\; if\ f(x) \equiv 0 \\ -2^n \;,\; if\ f(x) \equiv 1 \\ \;\;0 \;\;\;,\; if\ f(x)\ is\ balanced \end{cases}} |^2 = \begin{cases} 1 \,,\; if\ f\ \ is\ constant \\ 0 \,,\; if\ f\ \ is\ balanced \end{cases}$$

If our function f is a constant function we will have our output values all zeros or our constant function will always map to one. In the first case of being constant, our function always gives 0 as the result so the $(-1)^{f(x)}$ value will be +1 then $\sum_{x \in \{0,1\}^n} (-1)^{f(x)}$ will equal to $+2^n$. If our function is constant and always gives the value 1 as the output our calculation $(-1)^{f(x)}$ will equal to -1 therefore $\sum_{x \in \{0,1\}^n} (-1)^{f(x)}$ will give the result -2^n.

However, if our function is a balanced function meaning half of the outputs of our function is 0 our calculation will equal $+2^{n-1}$, and the other half will be 1 therefore our calculation will be -2^{n-1} . Adding those two values will cancel each other so we will have 0 as our result if our function is a balanced one.

Introduction to Grover’s Algorithm

Devised by Lov Grover in 1996[1], Grover’s Algoirthm searches through an unstructured database with a superior speed than a classical algorithm to find the given values. The problem in Grover’s Algorithm is to find spesific values among an unsorted database with the least amount of time using amplitude amplification.

Grover’s algorithm is the first algorithm proven to be faster than classical algorithms solving the same problem.

GROVER’S ALGORITHM

Algorithms “searching an unsorted database” with $N = 2^n$ elements in $O(\sqrt{n})$ time

Rather find x such that $f(x) = 1$

Classical Algorithm needs on average $\frac{N}{2} = O(N)$ time

There is no better classical algorithm

<u>GOAL</u>: We want to find our winner element w, we are given an oracle U_f which implements the function $f: \{0,1\}^n \rightarrow \{0,1\}$ which takes n input bits and outputs either 0 or 1. We define $f(x)$ such that it outputs 1 if the input value x is the winner element (w) and outputs 0 if its not the winner element. We also need one other function $f_0(x)$ which is independent of w, and it outputs 0 if the input was 0 bit-string, and 1 if it was anything other than 0 bit-string

$$f(x) = \begin{cases} 1\,, if\ x\ =\ w \\ 0\,, if\ x\ \neq w \end{cases}$$

The role of U_f is that if applied it would give -1 to the $f(x)$ times the state $|\,x\rangle$. When we define our Phase oracle U_f, and we want to output $|\,w\rangle$ as $-|\,w\rangle$ and every other element $|\,x\rangle$ should output to itself $|\,x\rangle$.

Phase oracle: $U_f\ |\,x\rangle\ =\ (-1)^{f(x)} =|\,x\rangle$

$$\boxed{\begin{aligned} U_f\ :&\ |\,w\rangle\ \rightarrow -|\,w\rangle \\ &\ |\,x\rangle\ \rightarrow\ |\,x\rangle\ \ \forall\ x \neq w \end{aligned}}$$

We can write this behavior as identity matrix minus 2 times the projection onto w . Lets try it, if we apply U_f onto $|w\rangle$ we will get identity times $|w\rangle$ which is $|w\rangle$ itself, and minus 2 $|w\rangle$ which will result in $-|w\rangle$ as predicted. If we apply a state $|x\rangle$ orthogonal to $|w\rangle$ we will have the same input $|x\rangle$ as the result.

$$U_f = \mathbf{1} - 2 \cdot |w\rangle\langle w|$$

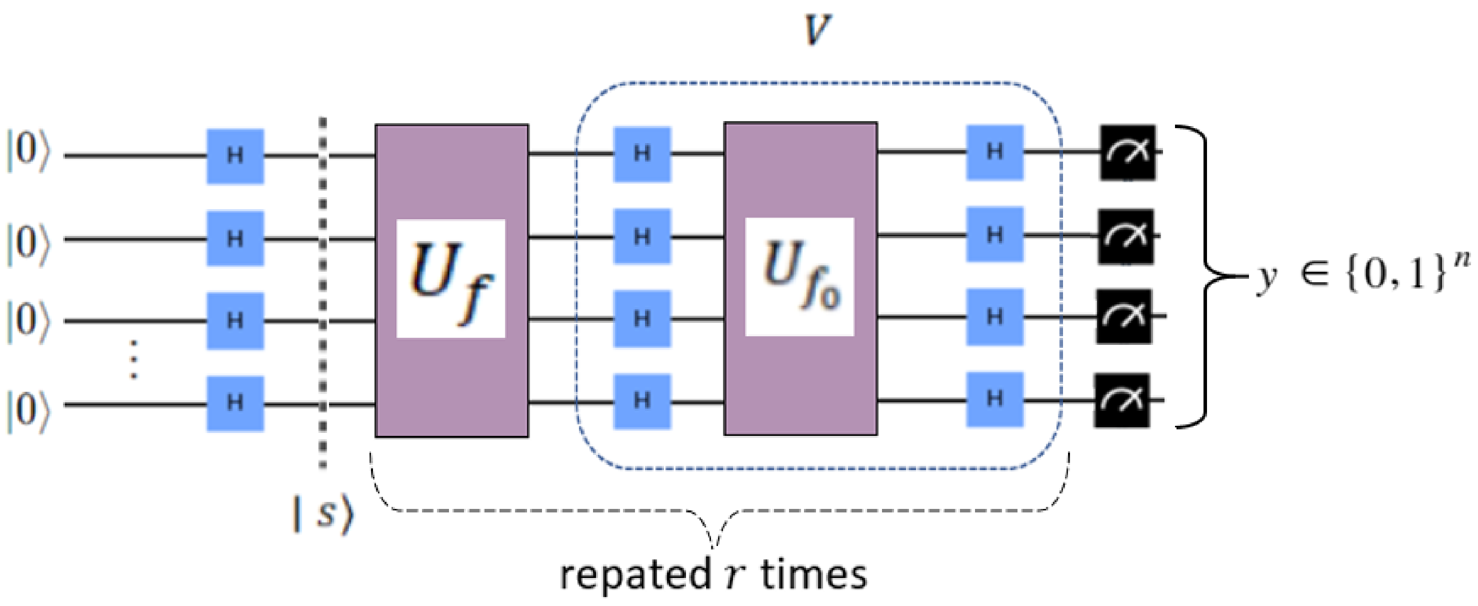
We can do a similar of that to function U_{f_0} , where we have our input $|0\rangle^{\otimes n}$ which should map to itself $|0\rangle^{\otimes n}$. And the input state $|x\rangle$ which is not the 0 bit-string, it will map to $-|x\rangle$. We can show this behavior as 2 times the projection onto the 0 bit-string minus the identity matrix.

$$\boxed{\begin{aligned} U_{f_0} : |0\rangle^{\otimes n} &\to |0\rangle^{\otimes n} \\ |x\rangle &\to -|x\rangle \quad \forall\ x \neq 00\ldots0 \end{aligned}}$$

$$U_{f_0} = 2 \cdot |0\rangle\langle 0|^{\otimes n} - \mathbf{1}$$

<u>**Quantum Circuit**</u>:

The quantum circuit for Grover's Algorithm consists of Hadamard Gates, U_f , U_{f_0} and the measurements at the end. We call the classical output of this circuit the $y\ bit - string$. The section that we call V is the <u>diffuser</u> in the algorithm, and the parts enclosed by the bracket will run r times in total



Claim: Our claim is that y, the output bits, will be equal to w with high probability.

$y = w$ (with very high probaility)

Proof:

Let us define the uniform superposition state. Which is the state after applying Hadamard Gates to all $|0\rangle$ qubits. We get the equal superposition of all 2^n states. This state is called $|s\rangle$.

$$|s\rangle = H^{\otimes n} |0\rangle^{\otimes n} = \frac{1}{\sqrt{2^n}} \sum_{x \in \{0,1\}^n} |x>$$

We define V as U_{f_0} and Hadamard Gates on its both sides. Here we will write U_{f_0} as its unitary representation, and we will write $|s\rangle$ in the places that are $H^{\otimes n} |0\rangle^{\otimes n}$. This simplifies to 2 times the projection onto s minus the identity matrix.

$$V = H^{\otimes n} \cdot U_{f_0} \cdot H^{\otimes n} = H^{\otimes n} \cdot 2 |0\rangle\langle 0|^{\otimes n} \cdot H^{\otimes n} - H^{\otimes n} \cdot H^{\otimes n}$$

$$= 2 |s\rangle\langle s| - \mathbf{1}$$

Grover's algorithm then carries out the operation $\left(V \cdot U_f\right)^r$ on the state $|s\rangle$. The reason why U_f is on the right of the expression is because we write from right to left so our starting point is always the right side.

Let $\sum$ be the plane spanned by $|s\rangle$ (the superposition state) and $|w\rangle$ (the winner element). Let $|w^{\perp}\rangle$ to be the state orthogonal to $|w\rangle$ and lies in this plane $\sum$ Do not forget that there are infinetely many states that are orthogonal to $|w\rangle$ but we want the orthogonal state that lies in this plane. So that we can write it as linear combination.

The equation for $| w^{\perp} >$ is shown here as the superposition of all states in $| s >$ except for $| w >$, which is the reason behind the -1 in the normalization factor of $2^n - 1$:

$$|w^{\perp}> = \frac{1}{\sqrt{2^n - 1}} \sum_{x \neq w} |x>$$

We can write $| s >$ in terms of $| w^{\perp} >$ and $| w >$, and when we sum these 2 states up we get the equal superposition of all states:

$$| s >= \sqrt{\frac{2^n - 1}{2^n}} \ | w^{\perp} > \ + \frac{1}{\sqrt{2^n}} \ | w >$$

Now we define an angle θ such that it equals $\sin\frac{\theta}{2} = \frac{1}{\sqrt{2^n}}$.
Since $\sin^2 = 1 - \cos^2$ we can write the other part as $\cos$. From there we can see what θ equals to:

$$\theta = 2 \cdot \sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)$$

$$| s >= \sqrt{\frac{2^n - 1}{2^n}} \ | w^{\perp} > \ + \frac{1}{\sqrt{2^n}} \ | w > \ = \ \cos\frac{\theta}{2} \, | w^{\perp} > + \sin\frac{\theta}{2} \, | w >$$

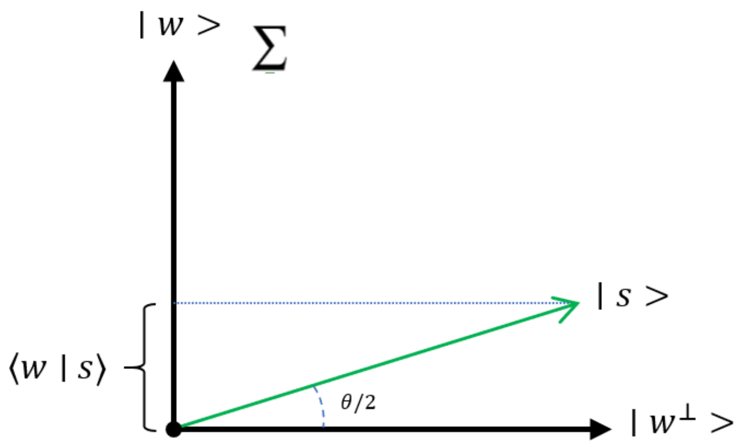
PROTOCOL FOR THE GROVER'S ALGORITHM:

We plot our plane $\sum$

In plane $\sum$ we have $| w >$ and $| w^{\perp} >$, and our state $| s >$

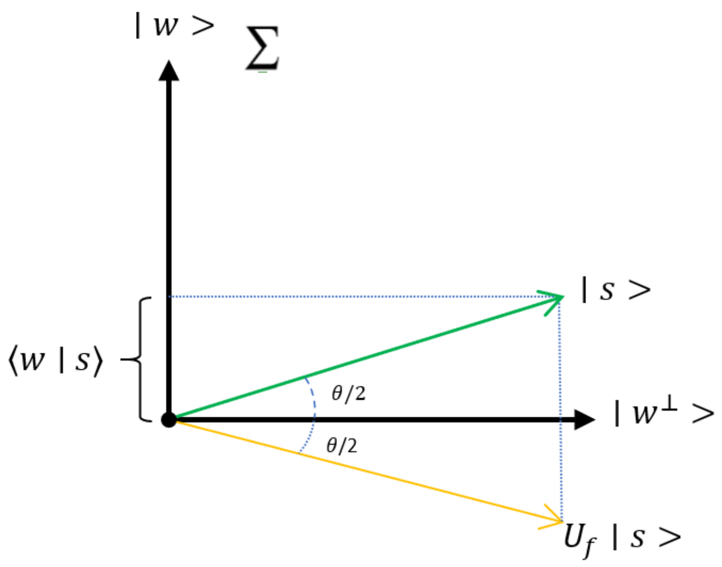
1. STEP : Prepare $| s >$

Since $| s >$ is the superposition of all states other than $| w >$, it is very close to $| w^{\perp} >$ in the plane.



2. STEP : Apply $U_f = \mathbf{1} - 2 \cdot | w \rangle\langle w | \rightarrow$ reflection at $| w^{\perp} >$

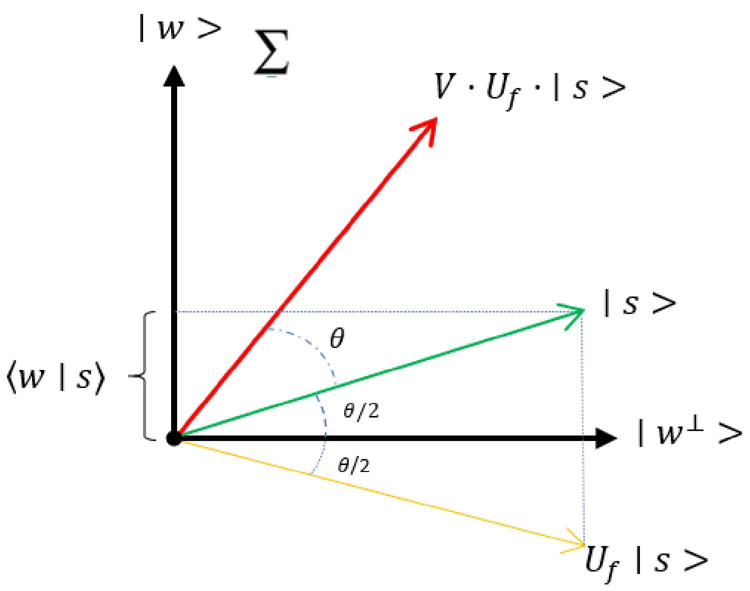
Since the following operation we have in the circuit is U_f , we are applying it to the state $| s >$. U_f applied to $| s >$ means that we have -2 times the projection onto $| w >$ which is a reflection at $| w^{\perp} >$.



3. STEP : Apply $V = 2 \mid s\rangle\langle s \mid -\mathbf{1}$ $\rightarrow$ reflection at $\mid s >$

We reflect $U_f \mid s >$ through the state $\mid s >$

$V \cdot U_f$ corresponds to a rotation by angle θ



In the Quantum Circuit for Grover's Algorithm we have r iterations of $V \cdot U_f$.
After r applications of step 2 and step 3 , the state $\mid s >$ is rotated by $r \cdot \theta$

We cannot rotate past $\frac{\pi}{2}$ degrees in total because cannot simply reach $\mid w >$.
Even though we minimize the amplitudes of other states, we can never have them equal to 0 therefore we can never rotate $\frac{\pi}{2}$ degrees or more. We choose r, such that $r \cdot \theta + \frac{\theta}{2} \approx \frac{\pi}{2}$. That is because we want to get as close as we can to the winner state $\mid w >$. If we solve this for r :

$$r = \frac{\pi}{2\theta} - \frac{1}{2} = \frac{\pi}{4 \cdot \sin^{-1}\left(\frac{1}{\sqrt{2^n}}\right)} - \frac{1}{2} \approx \frac{\pi}{4}\sqrt{2^n} = O(\sqrt{N})$$

We already defined θ and we plug that in. If n is very large the $arcsin$ of x will be x. This scales as $\sqrt{N}$ and our big O notation is $O(\sqrt{N})$.

After r calls to the oracle, the final measurement will result in state $| w >$ with minimum probability:

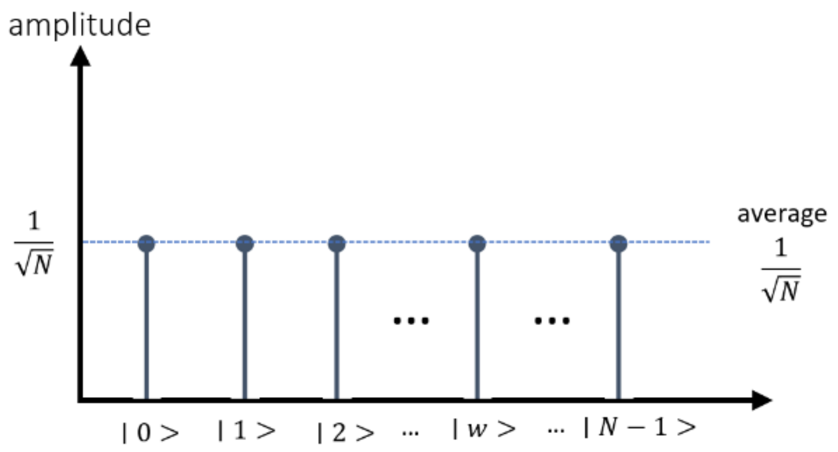
$$p(w) \geqslant \mathbf{1} - \sin^2 \frac{\theta}{2} = 1 - \frac{1}{2^n}$$

APLITUDE AMPLIFICATION

The main idea behind Grover's Algorithm is amplitude amplification. Let us have a look at the amplitudes at each step in Grover's Algorithm:

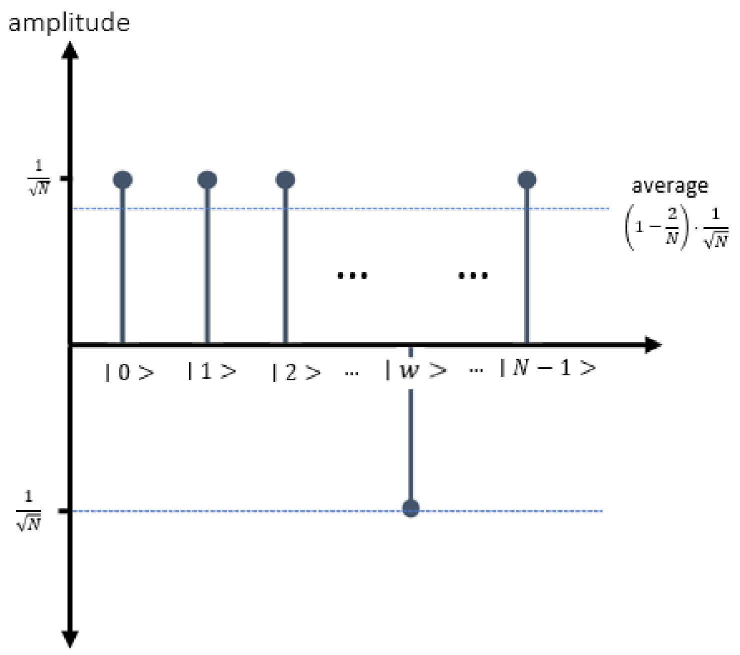
1. STEP : $| s > = H^{\otimes n} \; | 0 >^{\otimes n}$

First we create our equal superposition state. Amplitudes of all states are same and equal to $\frac{1}{\sqrt{N}}$. All states have thse same probability.

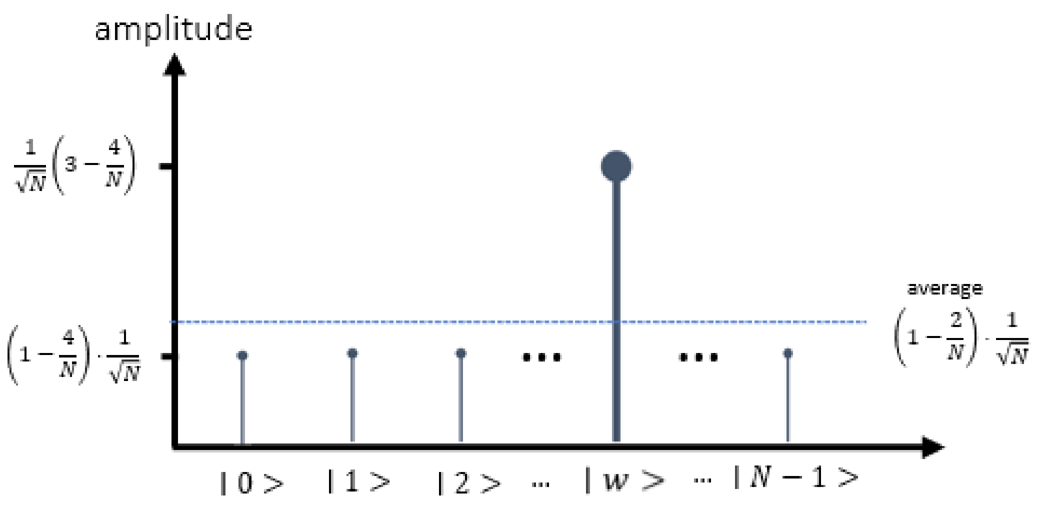


2. STEP: $U_f \mid s> \; = \; (1 - 2 \cdot \mid w\rangle\langle w \mid) \mid s> \; \rightarrow$ flip the amplitude of $\mid w\rangle$

This state is when we apply our oracle U_f which flips the amplitude of our winner state $\mid w\rangle$ to - $\frac{1}{\sqrt{N}}$. The other elements do not have any change in this step. As a result the average drops to $\left(1 - \frac{2}{N}\right) \cdot \frac{1}{\sqrt{N}}$



3.STEP : $V \cdot U_f \cdot \mid s> \; = \; (2 \mid s\rangle\langle s \mid -\mathbf{1}) \cdot \; U_f \; \cdot \mid s> \; \rightarrow$ reflect the amplitudes about the average amplitude. In this step we are applying our operator V which is called the diffuser. All amplitudes in this step are reflected through the average amplitude.



Reflection of α_j about the average $< \alpha >$

By repeating the steps 2 & 3 , the amplitude of $| w\rangle$ will increase further
=> amplitude amplification. When we have a higher amplitude for $| w\rangle$ we have a higher probability for measuring it.

MULTIPLE MARKED (WINNER) ELEMENTS

In some problems there may be more than one solution at the end. When we have M marked winner elements w_i , we define the winning state as a superposition of all the elements. Our $| w^{\perp} >$ elements has a different normalization factor because now we have M elements that are not included in $| w^{\perp} >$. Instead of $N - 1$ now we have $N - M$ part of the normalization factor.

$$| w >= \frac{1}{\sqrt{M}} \sum_{i=1}^{M} | w_i >$$

$$| w^{\perp} >= \frac{1}{\sqrt{N - M}} \sum_{x \neq \{w_{1,..,}w_m\}}^{M} | x >$$

If we write $| s >$ as the sum of $| w^{\perp} >$ and the winner states $| w >$ our normalization factors change accordingly. We can define θ again such that we can write the expression in terms of sin and cos .

$$| s >= \frac{\sqrt{N - M}}{\sqrt{N}} \cdot | w^{\perp} > + \sqrt{\frac{M}{N}} \cdot | w >$$

$$= \cos\left(\frac{\theta}{2}\right) | w^{\perp} > + \sin\left(\frac{\theta}{2}\right) | w >$$

The rest of the algorithm works like the situation where we had one winner element $| w >$ but now the angle θ changed. It was $\sqrt{\frac{1}{N}}$ but now it is $\sqrt{\frac{M}{N}}$ so that we a greater angle θ.

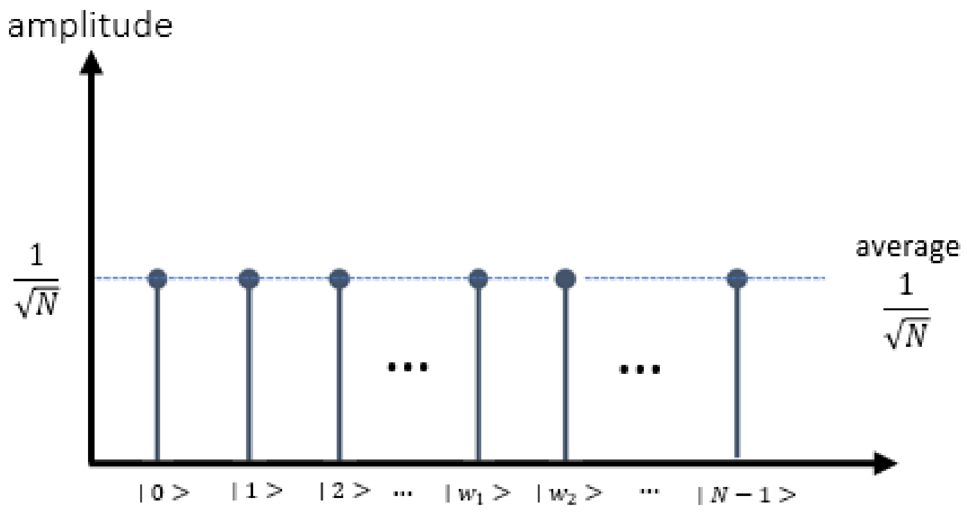
$\sin\left(\frac{\theta}{2}\right) = \sqrt{\frac{M}{N}}$ $\rightarrow$ the angle becomes larger

Since we rotate by an angle θ each time and we have M winner elements in the total N states we will have our results faster. When determining r we can write the following expression and see that our runtime will be smaller.

$$r = \frac{\pi}{4 \cdot \sin^{-1}\left(\sqrt{\frac{M}{N}}\right)} - \frac{1}{2} = O\left(\sqrt{\frac{N}{M}}\right)$$

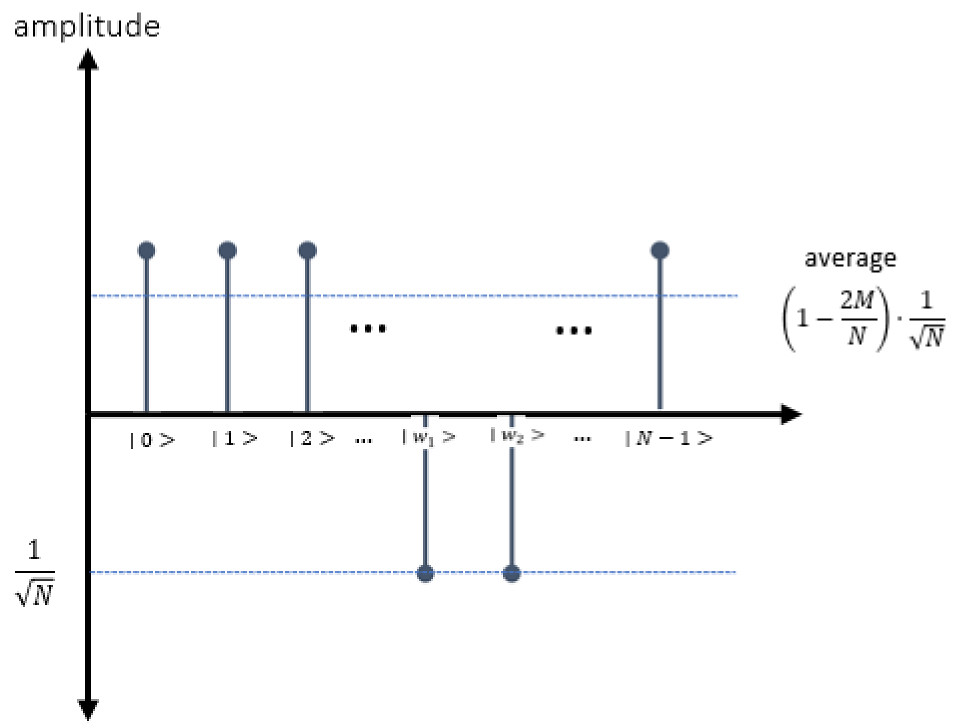
We can see this step also looking at the amplitudes:

In the first step all states are in a superposition state

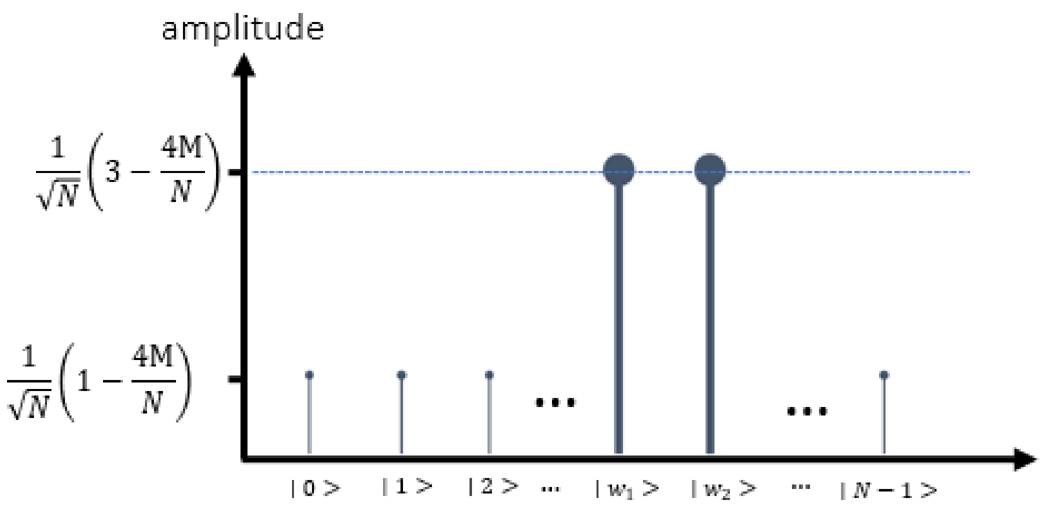


In the second step we are flipping all the winner states w_i just like when we had one winner element. When M is smaller the average amplitude will be smaller, so when the number of winner states M increases the average of all states decreases.

Since we reflect the winner states w_i through the average we will have a greater amplitude difference than when we had one winner state $| w >$



For the last step when we reflect the $| w_i >$ about the average over and over again the average in each iteration will decrease further leaving us with greater amplitude differences between winner states and other elements. Which results in a faster calculation.


amplitude
$\frac{1}{\sqrt{N}}\left(3-\frac{4M}{N}\right)$
$\frac{1}{\sqrt{N}}\left(1-\frac{4M}{N}\right)$
$|0>$
$|1>$
$|2>$
$|w_1>$
$|w_2>$
$|N-1>$

CONCLUSIONS

With the advancements in Physics and Computation, Quantum mechanics provided new ways to perform computations. There are a wide range of problems where quantum approaches are exponentially faster solving a given computational problem than classical machines. Deutsch-Jozsa and Grover’s Algorithms are two of the initial algorithms introduced to the field, and that two algorithms give an insight for the potential of using Quantum Algorithms. Researching the algorithms and understanding how they work in detail allowed us to understand the story behind the advantage they have. In this paper, we looked through detailed breakdowns of Deutsch-Jozsa and Grover’s Allgorithms, including a side-by-side comparison of the quantum algorithm and its classical counterpart. The future of this field offers significant andvancements in computation power of mavhines and Quantum Computers will continue to outperfom Classical Computers in solving complex computational problems, such as simulations, making use of the quantum phenomena.

ACKNOWLEDGEMENT

I would like to thank my mentor, Daniel Kolodrubetz for his guidance and help throught this paper.

REFERENCES

L. K. Grover (1996), "A fast quantum mechanical algorithm for database search", Proceedings of the 28th Annual ACM Symposium on the Theory of Computing (STOC 1996), doi:10.1145/237814.237866, arXiv:quant-ph/9605043

I. Chuang & M. Nielsen, "Quantum Computation and Quantum Information", Cambridge: Cambridge University Press, 2000.

Grover L.K.: *From Schrödinger's equation to quantum search algorithm*, American Journal of Physics, 69(7): 769–777, 2001. Pedagogical review of the algorithm and its history.

Team, The Qiskit. "Grover's Algorithm." Qiskit. Data 100 at UC Berkeley, June 22, 2021. https://qiskit.org/textbook/ch-algorithms/grover.html#Amplitude-Amplification.

David Deutsch and Richard Jozsa (1992). "Rapid solutions of problems by quantum computation". Proceedings of the Royal Society of London A. 439: 553–558. doi:10.1098/rspa.1992.0167.

R. Cleve; A. Ekert; C. Macchiavello; M. Mosca (1998). "Quantum algorithms revisited". Proceedings of the Royal Society of London A. 454: 339–354. doi:10.1098/rspa.1998.0164.

- Qiskit summer school lectures

https://en.wikipedia.org/wiki/Toy_problem#:~:text=In%20scientific%20disciplines%2C%20a%20toy,a%20particular%2C%20more%20general%2C%20problem

https://en.wikipedia.org/wiki/Ancilla_bit#:~:text=In%20quantum%20computing%2C%20quantum%20catalysis,bits%20for%20quantum%20error%20correction.

https://docs.microsoft.com/en-us/azure/quantum/concepts-oracles

https://en.wikipedia.org/wiki/Deterministic_algorithm

https://qiskit.org/textbook/ch-algorithms/deutsch-jozsa.html

https://www.physics.umd.edu/courses/Phys374/fall05/files/DiracNotation.pdf

https://qiskit.org/learn/intro-qc-qh/