CSE 599d - Quantum Computing
Reversible Classical Circuits and the Deutsch-Jozsa Algorithm

Dave Bacon
Department of Computer Science & Engineering, University of Washington

Now that we've defined what the quantum circuit model, we can begin to construct interesting quantum circuits and think about interesting quantum algorithms. We will approach this task in basically an historical manner.

A few lectures ago we saw Deutsch's algorithm. Recall that in this algorithm we were able to show that there was a set of black box functions for which we could only solve Deutsch's problem by querying the function twice classically, but when we used the boxes in their quantum mode, we only needed to query the function once. Now this was certainly exciting. But one query versus two query is a lot like getting half price for a matinee. Certainly it is a great way to save money, but its no way to build a fortune (Yeah, yeah, you can go on all you want about Ben Franklin, but a penny saved is most of the time a waste of money these days.) In the coming lectures we will begin to explore generalizations of the Deutsch algorithm to more qubits. For it is in the scaling of resources with the size of the problem that our fortunes will be made (or lost.)

I. REVERSIBLE AND IRREVERSIBLE CLASSICAL GATES

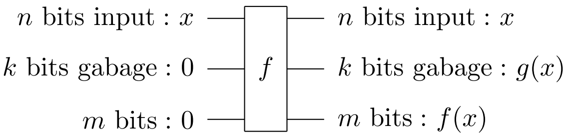
Before we turn to the Deutsch-Jozsa algorithm, it use useful to discuss, before we begin, reversible versus irreversible classical circuits. A classical gate defines a function f from some input bits $\{0,1\}^n$ to some output bits $\{0,1\}^m$. If this function is a bijection, then we say that this circuit is reversible. Recall that a bijection means that for every input to the function there is a single unique output and for every output to the function there is a single unique input. Thus given the output of a reversible function, we can uniquely construct its input. A gate which is not reversible is called irreversible. Note that our definition of reversible requires $n = m$. Why consider reversible gates? Well one of the original motivations comes from Landauer's principle. So it is useful to make a slight deviation and discuss Landauer's principle.

Suppose that we examine a single bit in our computer. This bit is a degree of freedom with two possible states. Now the rest of the computer has lots of other degrees of freedom and can be in one of many states. Now one of the fundamental theorems of physics says that phase space cannot be compressed. Lets apply this principle to the process of erasing our bit. Suppose there is a physical process which irreversibly erases a bit: 1 goes to 0 and 0 goes to 0. Now sense phase space cannot be compressed, if we perform this irreversible operation, then the phase space must "bulge out" along the other degrees of freedom. In particular this implies that the entropy of the those other degrees of freedom must increase. If you work through how much it must increase, you will find that it must increase by $k_BT\ln 2$ for a system in thermodynamic equilibrium. This is Landauer's erasure principle: erasing a single bit of information necessarily results in the increase in entropy of the other degrees of freedom by a value of at least $k_bT\ln 2$.

Landauer's erasure principle motivates us to consider reversible computation. Now today's modern computers are not near Landauer's limit: they dissipate something like a five hundred times as much heat as Landauer's limit. But certainly there are expectations that this will change as we build smaller and smaller computers. Now one thing you might worry about is whether the power of reversible computation is the same as the power of irreversible computation. And this is a great question. I won't go into all of the details, but there are results which basically say that they have the same computational power. But it is nice to see, at this point, a trick which we can use to compute irreversible functions using reversible circuits.

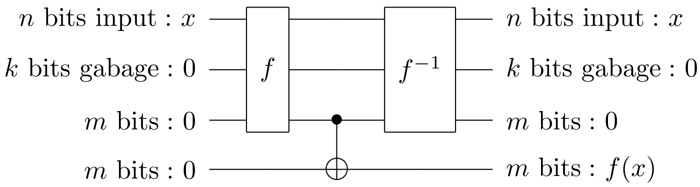
The trick is as follows. Suppose that we have a irreversible gate f from n bits to m bits. Then we can design a reversible gate which computes this same function: we just keep the input around. In particular we can construct a reversible gate with $n+m$ bits input which acts as $(x,y) \rightarrow (x, y \oplus f(x))$ where the $\oplus$ is the bitwise XOR. It is easy to see that this is reversible: simply apply it twice: $(x,y) \rightarrow (x, y\oplus f(x)) \rightarrow (x, (y\oplus f(x)) \oplus f(x)) = (x,y)$, so the function is its own inverse and is defined over all possible inputs. Further if we apply this reversible circuit element to $(x,0)$ we obtain $(x, f(x))$, i.e. we have computed $f(x)$. Thus we can always turn a irreversible gate into a reversible gate, at the expense of having a larger gate: a gate with "garbage bits" which aren't necessarily used in the irreversible circuit.

Now when we get to quantum circuits for these reversible gates we will see that these garbage bits can have bad effects on our quantum algorithms. So if we want to simulate the action of a irreversible circuit from n bits input to m bits output, is there a way to get rid of these garbage bits? There is! This is a beautiful trick due to Charlie Bennett in 1973. Suppose that we have a function which takes the n bits as input and is implemented using our above construction which produces garbage bits. This will be a circuit like



(1)

Now here is the trick to get rid of the garbage bits. Since f is made up of reversible circuits, we can implement f^{-1} by simply reversing the order of all of the circuit elements and then implementing the inverse of each of these elements. Further note that the controlled-NOT is a reversible gate and can be used, on a 0 target, to copy the value of a bit to the target bit. Using these two facts we can implement the following circuit:



(2)

Where the controlled-NOT is really m bitwise controlled-NOTs. Thus it is always possible to erase all of the garbage bits. This trick will be used all the time, often implicitly.

Why are reversible gates important? Because, as we saw in Deutsch's algorithm, we can turn these gates into quantum gates which enact the reversible computation on the computational basis. In particular if $h : \{0,1\}^n \rightarrow \{0,1\}^n$ is a reversible gate, we can implement it as

$$\sum_{x\in\{0,1\}^n} |h(x)\rangle\langle x| \tag{3}$$

Since $h(x)$ is reversible, this will be a unitary gate on n qubits and, on the computational basis, it will compute the reversible function h. Using the garbage trick we can implement a general irreversible classical circuit by changing the irreversible circuit elements to the quantum reversible circuit elements and then use the garbage trick to enact a quantum circuit which enacts the unitary evolution

$$\sum_{x=0}^{2^n-1}\sum_{y=0}^{2^m-1} |x\rangle\langle x| \otimes |y \oplus f(x)\rangle\langle y| \tag{4}$$

What I've just discussed, that we can turn irreversible gates into reversible gates and these reversible gates into quantum gates, along with the fact that, at the cost of a factor of 2 we can get rid of garbage bits, is used implicitly in quantum computing all the time.

One further interesting question which is useful to know something about is universal sets of gates for classical computing. Just as we discussed sets of gates which can be used to compute any unitary operation on our quanatum data, we can discuss gates which can be used to implement any reversible or irreversible circuit. A famous example of a irreversible gate which is universal is the NAND (not AND) gate. This is a two bit gate which computes $f(0,0) = 1, f(0,1) = 1, f(1,0) = 1, f(1,1) = 0$. An interesting question is whether there are any classical reversible gates with two bits input and output which is universal. The answer is no! The smallest gate which is universal for reversible classical circuits requires three bits input and output.

For example, let's prove that the controlled-NOT gate, by itself, can never be used for universal classical computation. For inputs $x, y \in \{0,1\}$, the controlled-NOT computes $(x,y) \rightarrow (x, x+y)$ where the addition is done modulo 2. Now suppose that we have n input bits $x_1, \dots, x_n$. A controlled-NOT with control on the ith bit and a target on the jth bit can be represented by an $n \times n$ matrix which has ones along its diagonal, and zeros everywhere else except on in the ith row and jth column which has a one. Then we can check that if we represent the input to the controlled-NOT by the vector $[x_1\ x_2\ \cdots\ x_n]^T$ then multiplication of this matrix working with addition modulo 2 is the same as enacting a controlled-NOT. Thus we see that by multiplying representations of the controlled-NOT together, and doing our addition modulo 2, we can represent the action of any series of controlled-NOTs on n bits by a $n \times n$ matrix. Now this means that there are at most 2^{n^2} such 0-1 matrices which can be realized by controlled-NOTs (there are actually less because not all $0-1$ matrices correspond to a controlled-NOT, for example the all zeros matrix.) But now think about what would be needed for a universal reversible circuit. If we look at the set of reversible circuits on n bits input, then this set corresponds exactly to the set of $2^n \times 2^n$ permutation matrices. There are $(2^n)!$ such functions. Remember that $x!$ grows like $\sqrt{x}\left(\frac{x}{e}\right)^x$. Thus we see that there is certainly no way that a controlled-NOT can be used on n qubits to reproduce all of these permutations. Further we can also see that even if we "encode" (for some reasonable definition thereof) the space of controlled-NOT reachable operations will never grow fast enough to keep up with the space of reversible circuits. Thus we see that controlled-NOTs can never be used as a universal gate set for reversible classical circuits. Similar constructions can be used to show that no possible two bit gate can be universal.

So what sorts of gates are universal for classical reversible computation. Two of the most famous are the Toffoli gate and the Fredkin gate. The Toffoli gate is the controlled-controlled-NOT. It enacts the transformation

$$\begin{aligned} 000 &\rightarrow 000 \\ 001 &\rightarrow 001 \\ 010 &\rightarrow 010 \\ 011 &\rightarrow 011 \\ 100 &\rightarrow 100 \\ 101 &\rightarrow 101 \\ 110 &\rightarrow 111 \\ 111 &\rightarrow 110 \end{aligned} \tag{5}$$

The Fredkin gate is the controlled-SWAP gate. It's truth table is

$$\begin{aligned} 000 &\rightarrow 000 \\ 001 &\rightarrow 001 \\ 010 &\rightarrow 010 \\ 011 &\rightarrow 011 \\ 100 &\rightarrow 100 \\ 101 &\rightarrow 110 \\ 110 &\rightarrow 101 \\ 111 &\rightarrow 111 \end{aligned} \tag{6}$$

Both the Toffoli and the Fredkin gates are universal for classical reversible computation, given the ability to use extra scratch bits in fixed states. An interesting further property of the Fredkin gate is that it is conservative: it never changes the number of 0's and 1s in a string. One reason to consider this gate is motivated by physics. Suppose that the 0 bit and the 1 bit are represented by configurations with different energies. Then with a logic gate like Fredkin's gate it is possible to conserve the energy of these bits from input to output. With a gate like Tofolli's gate, this would not be possible if the bits have different energies. It is interesting to go through the current physical implementations of bits and to note whether the bits in these devices are implemented in configurations with different energies.

II. THE DEUTSCH-JOZSA ALGORITHM

Now that we've discussed reversible classical computation, we can move on to the quantum algorithms.

We have already discussed Deutsch's algorithm, and seen how we could distinguish between two different classes of functions using one single quantum query, while if we stuck to the classical world, we needed at two queries. In 1992, Deutsch and Jozsa generalized this result. In the Deutsch-Jozsa problem, the two classes of functions we wish to distinguish are the *constant* and *balanced* functions. Consider a function from n bits to 1 bit. We call the function constant if the function is the same on all inputs (the function is either $f(x) = 0$ or $f(x) = 1$.) We call the function balanced if the function is zero on half of the possible inputs and one on the other half of the possible inputs ($f(x) = 0, \forall x \in S$ and $f(x) = 1, \forall x \neq S$ and $|S| = 2^{n-1}$.) The problem Deutsch and Jozsa considered was how many queries it would take in the worst case to distinguish between whether the function is constant or whether it is balanced.

Let's analyze how hard the Deutsch-Jozsa problem is. To do this we perform a worse case analysis of the query complexity. Suppose that we are querying the function for multiple different inputs x_i. Now it is clearly not useful to query the same value twice. After querying k times, we will have learned something about the function. But note that in the worst case if $k \leq 2^{n-1}$ the value of the function queried may all be the same. And in this case we cannot determine whether the function was constant or whether it was balanced. Thus in the worst case, we need $k = 2^{n-1} + 1$ queries to distinguish whether the function was constant or balanced.

Now what happens in the quantum world? Well we need to recast the function we were querying, which was a irreversible function, as a reversible function. Thus we assume in the quantum model, that we have a black box which implements the function in a reversible manner as

$$U_f = \sum_{x\in\{0,1\}^n} \sum_{y\in\{0,1\}} |x\rangle\langle x| \otimes |y \oplus f(x)\rangle\langle y| \tag{7}$$

Note that if we query this function using computational basis states, like $|x, y\rangle$, then we obtain the value $|x, y \oplus f(x)\rangle$. If $y = 0$ this means we have computed $f(x)$. If $y = 1$ we have computed $\bar{f}(x)$, the negation of $f(x)$. Either way, when we do this, we really only obtain information about $f(x)$, so our analysis of the query complexity if we only input computational basis states is exactly that of our classical analysis above.

OK, so here is our quantum game. We wish to query the function using U_f and distinguish whether the function f is constant or balanced using as few queries as possible.

The first trick we'll use to do this is called phase kickback. Suppose that we input into the second register of U_f the state $|-\rangle = \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. To see how this acts on the n input bits to U_f, we calculate how this unitary acts on $|x\rangle \otimes |-\rangle$

$$U_f|x\rangle \otimes |-\rangle = \left(\sum_{x'\in\{0,1\}^n} \sum_{y'\in\{0,1\}} |x'\rangle\langle x'| \otimes |y' \oplus |f(x') \oplus y'\rangle\langle y'| \right) |x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle) \tag{8}$$

which is equal to

$$U_f|x\rangle \otimes |-\rangle = \frac{1}{\sqrt{2}} \left(|x\rangle \otimes |f(x)\rangle - |x\rangle \otimes |\bar{f}(x)\rangle \right) = |x\rangle \otimes \frac{1}{\sqrt{2}}(|f(x)\rangle - |\bar{f}(x)\rangle) \tag{9}$$

Now if $f(x) = 0$, then this is $|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$ and if $f(x) = 1$, then this is $|x\rangle \otimes \frac{1}{\sqrt{2}}(|1\rangle - |0\rangle) = -|x\rangle \otimes \frac{1}{\sqrt{2}}(|0\rangle - |1\rangle)$. Thus we have shown that

$$U_f|x\rangle \otimes |-\rangle = (-1)^{f(x)}|x\rangle \otimes |-\rangle. \tag{10}$$

This is the *phase kickback* trick. By querying the register where the function is computed in a superposition of $|0\rangle$ and $|1\rangle$, we can kick back the value of the function into the phase of the input register.

Suppose that we input into the input register of U_f as superposition over all possible input strings and $|-\rangle$ into the register where the function is computed. In other words suppose that we query U_f with the input

$$|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \otimes |-\rangle \tag{11}$$

Then

$$U_F|\psi\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \otimes |-\rangle \tag{12}$$

Now this output is separable between the input qubits and the register where we compute the function. On the input qubits the state is now

$$\frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{f(x)}|x\rangle \tag{13}$$

Well have we gotten anywhere? Well we now have a superposition over all possible inputs to the function where the value of the function is computed into the phase of the amplitudes of different possible inputs. How do we use these inputs? Well first of all notice that if the function is constant, then the above state is an equal superposition over all n bit strings with a constant amplitude. If the function is balanced, then the above states is an equal superposition over all n bit strings with half of the amplitudes of positive sign and the other half being of negative sign. In particular one will note that states from this first case are *orthogonal* to the states from the second case (although within these two cases the states are not orthogonal.) This implies that we can, in principle, perform a measurement in a new basis which will allow us to distinguish with one hundred percent certainty which two sets we have been given.

In particular we can achieve this in a very simple manner. What we do is we apply the Hadamard gate to every qubit. Recall that the Hadamard gate is given by

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} = \sum_{x,y\in\{0,1\}} (-1)^{xy} |y\rangle\langle x| \tag{14}$$

Thus if we tensor this together n times, we obtain the gate

$$\begin{aligned} H^{\otimes n} &= \left(\frac{1}{\sqrt{2}} \sum_{x,y\in\{0,1\}} (-1)^{xy} |y\rangle\langle x| \right)^{\otimes n} \\ &= \left(\frac{1}{\sqrt{2}} \sum_{x_1,y_1\in\{0,1\}} (-1)^{x_1 y_1} |y_1\rangle\langle x_1| \right) \otimes \cdots \otimes \left(\frac{1}{\sqrt{2}} \sum_{x_n,y_n\in\{0,1\}} (-1)^{x_n y_n} |y_n\rangle\langle x_n| \right) \\ &= \frac{1}{\sqrt{2^n}} \sum_{x,y\in\{0,1\}^n} (-1)^{x\cdot y} |y\rangle\langle x| \end{aligned} \tag{15}$$

where $x \cdot y = x_1 y_1 + x_2 y_2 + \cdots + x_n y_n$. $H^{\otimes n}$ will be our good fried for the next few lectures, so we should get to know it pretty well. The first thing to know about $H^{\otimes n}$ is that it can be used to create an equal superposition over all possible bit strings:

$$H^{\otimes n}|0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \tag{16}$$

And, since $H^2 = I$, we also use this gate to take an equal superposition to the computational basis state $|0\rangle$:

$$H^{\otimes n} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle = |0\rangle \tag{17}$$

It is this latter property which we will use in the Deutsch-Jozsa algorithm.

Suppose that the function we are querying is constant. If we apply the Hadamard gates to the state produced after the phase kickback trick, then the new state will be

$$H^{\otimes n} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}} (-1)^{f(0)} |x\rangle = (-1)^{f(0)} H^{\otimes n} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}} |x\rangle = (-1)^{f(0)} |0\rangle \tag{18}$$

Thus if we measure in the computational basis after performing the n Hadamards, and the function is constant, then the measurement outcome will always be $|0\rangle$.

Now suppose, on the other hand, that the function we are querying is balanced. If we apply the Hadamard gate to the state produced after the phase kickback trick, then the new state will be

$$\begin{aligned} H^{\otimes n} \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}} (-1)^{f(0)} |x\rangle &= \frac{1}{2^n} \sum_{x',y'\in\{0,1\}^n} (-1)^{x'\cdot y'} |y'\rangle\langle x'| \sum_{x\in\{0,1\}^n} (-1)^{f(x)} |x\rangle \\ &= \frac{1}{2^n} \sum_{x,x',y'\in\{0,1\}^n} (-1)^{x'\cdot y' + f(x)} \delta_{x,x'} |y'\rangle = \frac{1}{2^n} \sum_{x,y'\in\{0,1\}^n} (-1)^{x\cdot y' + f(x)} |y'\rangle \end{aligned} \tag{19}$$

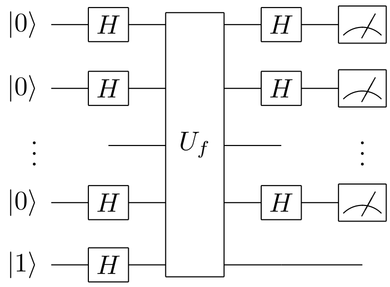
If we now measure in the computational basis, what is the probability that we will obtain the result $|0\rangle$? This is the magnitude squared of the amplitude of the above state along the $|0\rangle$ direction:

$$\left| \langle 0| \frac{1}{2^n} \sum_{x,y'\in\{0,1\}^n} (-1)^{x\cdot y' + f(x)} |y'\rangle \right|^2 = \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{x\cdot 0 + f(x)} \right|^2 = \left| \frac{1}{2^n} \sum_{x\in\{0,1\}^n} (-1)^{f(x)} \right|^2 \tag{20}$$

But notice that since $f(x)$ is balanced, i.e. it has an equal number of inputs which produce a zero and a one as output, then this sum must be zero. Hence we see that there is absolutely no probability that of observing the result $|0\rangle$.

Thus we have shown that by using the phase kickback trick, performing an n qubit Hadamard on the resulting quantum state, and measuring in the computational basis, then we can distinguish a constant function from a balanced equation using only a single quantum query: if the function is constant we will observe $|0\rangle$ and if the function is balanced we will never observer $|0\rangle$. This compares, rather favorably to the classical case, where in the worst case we required $2^{n-1}+1$ queries to distinguish constant functions from balanced functions. One versus exponential is pretty good!

Here is a quantum circuit for the Deutsch-Jozsa algorithm



(21)

Now a good question about the Deutsch-Jozsa algorithm is whether we should take the separation between a single quantum query and a required exponential number of queries seriously. Why? Well because in the real world, we might not require our classical algorithms to always succeed, but only that the classical algorithm succeed with small enough probability that our chances of failing are not relevant.

What are the implications of this for the Deutsch-Jozsa problem? Suppose that we query the function k times at randomly chosen different inputs. If we get the same outcome for this query, we answer that the function is constant and if we get different outcomes then we answer that the function is balanced. Note that the algorithm will only fail if the function is balanced (and we guess constant.) The probability of failing is bounded by the probability

$$p_{fail} \leq \frac{1}{2}^{k-1} \tag{22}$$

which shrinks exponentially in the number of queries. Thus for small k we can bound the probability of failing to be so small that it will never be of consequence. This means that when we allow a probability of failing, the Deutsch-Jozsa problem has a constant classical query complexity (Notice that it is important that the probability of failing does not grow with the instance size n.) Thus the separation we have achieved in the Deutsch-Jozsa problem is not as impressive as it was before (although there is still a constant separation.)

The above analysis is an example of how we deal with algorithms that can fail with some probability. Suppose that we solve a classical query complexity problem with an algorithm which fails with probability $\frac{1}{2}-\delta$: that is it gives us the wrong answer with probability $\frac{1}{2}-\delta$ and the correct answer with the larger probability $\frac{1}{2}+\delta$, where δ is a fixed constant. Now suppose that we run this algorithm k times and take the majority vote of the outcome of all of the algorithm. What is the probability of the algorithm failing? The probability of the algorithm succeeding j times and failing $k-j$ times is

$$\binom{k}{j}\left(\frac{1}{2}+\delta\right)^j\left(\frac{1}{2}-\delta\right)^{k-j} \tag{23}$$

Thus the probability of the algorithm failing is

$$p_f = \sum_{j=0}^{\lfloor \frac{k}{2} \rfloor}\binom{k}{j}\left(\frac{1}{2}+\delta\right)^j\left(\frac{1}{2}-\delta\right)^{k-j} \tag{24}$$

The largest term in this sum occurs when $j=\lfloor\frac{k}{2}\rfloor$. For this term, we note that the probability of the event associated with this term is

$$\left(\frac{1}{2}+\delta\right)^{\lfloor \frac{k}{2} \rfloor}\left(\frac{1}{2}-\delta\right)^{\lceil \frac{k}{2} \rceil} = \frac{(1-4\delta^2)^{\frac{k}{2}}}{2^k} \tag{25}$$

Now there are at most 2^n such terms in our probability of failing, so the probability of failing is

$$p_f \leq 2^k \times \frac{(1-4\delta^2)^{\frac{k}{2}}}{2^k} = (1-4\delta^2)^{\frac{k}{2}} \tag{26}$$

Finally recall that $1 - x \leq \exp(-x)$ so

$$p_f \leq e^{-2\delta^2 k} \tag{27}$$

Thus we see that the probability of failing falls exponentially in the number of times we repeat the algorithm. This bound we have derived is called a Chernoff bound.

In practical terms, what does the Chernoff bound mean? Suppose that the probability of our algorithm failing is 1/4 ($\delta = \frac{1}{4}$). Then here is the what the Chernoff bound tells us

k	$e^{-2\delta^2 k}$
10	≈ 0.29
100	$\approx 3.7 \times 10^{-6}$
500	$\approx 7.2 \times 10^{-28}$

Thus with only a small number of repetitions, we can make the probability of failing so small that it is much more likely that your computer will be hit by a meteor (actually the probability of being hit by a meteor integrated over your entire life is probably something like on in ten billion. Thus the probability in the 500 column is small indeed!)

Thus if we can bound the probability of our algorithm failing a fixed constant below $\frac{1}{2}$ we can effectively say that we have efficient solution for this problem. We may have to run the algorithm a hundred or a thousand times more, but this constant (independent of the problem size) isn't something that will bother us (which is not to say that such constants aren't extremely important!)

At this point it is useful to introduce the above notions into a fixed notation. We say that an algorithm is *exact* if it never fails. We say that an algorithm has *bounded error* if it fails with some probability. Note that we can apply these notions to both classical and quantum algorithms. Usually when we talk about bounded errors we fix some fixed probability for which the algorithm can fail, like say 1/4, but because of the Chernoff bound this exact number isn't too relevant. In this language, we can say that the Deutsch algorithm is an exact quantum algorithm with query complexity 1. Further the exact classical algorithm has a query complexity of $\Theta(2^{n-1}+1)$ and a bounded error query complexity of $\Theta(C)$ where C is some fixed small constant. Because we have an exact quantum algorithm, there is no need to discuss a bounded error quantum algorithm.

Acknowledgments

The diagrams in these notes were typeset using Q-circuit a latex utility written by graduate student extraordinaires Steve Flammia and Bryan Eastin.