1 Review

Recall that we can write any classical circuit $x \to f(x)$ as a reversible circuit R_f. We can view R_f as a unitary operation U_f, which acts on a quantum superposition using linearity. We will use this representation of a function throughout the lecture.

2 Phase State

We will first see how to set up an interesting state which we will use later in Fourier sampling. Recall the Deutsch-Jozsa procedure which, given a classical circuit for computing a boolean function $f : \{0,1\}^n \to \{0,1\}$, shows how to transform it into a quantum circuit that produces the quantum state $\left|\phi\right\rangle = 1/2^{n/2} \sum_x (-1)^{f(x)} \left|x\right\rangle$.

The quantum algorithm to carry out this task uses two quantum registers, the first consisting of n qubits, and the second consisting of a single qubit.

- Start with the registers in the state $|0^n\rangle\,|0\rangle$
- Compute the Fourier transform on the first register to get $\sum_{x\in\{0,1\}^n} |x\rangle \otimes |0\rangle$.
- Compute f to get $\sum_x |x\rangle\,|f(x)\rangle$.
- Apply a conditional phase based on $f(x)$ to get $\sum_x (-1)^{f(x)} |x\rangle\,|f(x)\rangle$.
- Uncompute f to get $\sum_x (-1)^{f(x)} |x\rangle \otimes |0\rangle$.

3 Fourier Sampling

Consider a quantum circuit acting on n qubits, which applies a Hadamard gate to each qubit. That is, the circuit applies the unitary transformation $H^{\otimes n}$, or H tensored with itself n times.

Applying the Hadamard transform (or the Fourier transform over Z_2^n) to the state of all zeros gives an equal superposition over all 2^n states

$$\mathscr{H}_{2^n} |0\cdots 0\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} |x\rangle \,.$$

In general, applying the Hadamard transform to the computational basis state $|u\rangle$ yields:

$$\mathscr{H}_{2^n} |u\rangle = \frac{1}{\sqrt{2^n}} \sum_{x\in\{0,1\}^n} (-1)^{u\cdot x} |x\rangle$$

We define the Fourier sampling problem as follows: Input an n qubit state $|\phi\rangle = \sum_{x\in\{0,1\}^n} \alpha_x |x\rangle$. Compute $H^{\otimes n}|\phi\rangle$ and measure the resulting state $\sum_y \hat{\alpha}_y |y\rangle$ to output y with probability $|\hat{\alpha}_y|^2$.

Clearly this problem is easy to solve on a quantum computer, but it appears to be hard to solve classically. We will see later by making the procedure recursive that Fourier sampling is indeed difficult in genereal.

4 Extracting n bits with 2 evaluations of Boolean Function

Suppose we are given a black box (or an obfuscated classical circuit) that computes the function function $f_s : \{0,1\}^n \to \{1,-1\}$, where $f(x) = s \cdot x$, where $s \cdot x$ denotes the dot product $s_1x_1 + \cdots + s_nx_n \bmod 2$. The challenge is to use this black box to efficiently determine s.

It is easy to see how to perform this task with n queries to the black box: simply input in turn the n inputs x of Hamming weight 1. The outputs of the black box are the bits of s. Since each query reveals only one bit of information, it is clear that n queries are necessary.

Remarkably there is a quantum algorithm that requires only two (quantum) queries to the black box:

- Use the black box to set up the phase state $|\phi\rangle = 1/2^{n/2} \sum_x (-1)^{f(x)} |x\rangle$.
- Apply the Fourier transform $H^{\otimes n}$ and measure. The outcome of the measurement is s.

To see that the outcome of the measurement is s, recall that $H^{\otimes n}|s\rangle = 1/2^{n/2} \sum_x (-1)^{s\cdot x}|x\rangle = |\phi\rangle$. Since $H^{\otimes n}$ is its own inverse, it follows that $H^{\otimes n}|\phi\rangle = |s\rangle$.

More generally, the transformation $H^{\otimes n}$ maps the standard basis $|s\rangle$ to the Fourier basis $|\phi_s\rangle = 1/2^{n/2} \sum_x (-1)^{s\cdot x}|x\rangle$ and vice-versa.

We have shown that a quantum algorithm can be more efficient than any probabilistic algorithm in terms of the number of queries. One way to use this difference in the number of queries in order to demonstrate a gap between quantum and probabilistic algorithms is to make the queries very expensive. Then the quantum algorithm would be $n/2$ times faster than any probabilistic algorithm for the given task. It turns out that we can increase this gap substantially, as we will see next. The idea is to make each query itself be the answer to a Fourier sampling problem, so each query itself is much easier for the quantum algorithm than for any probabilistic algorithm. Carrying this out recursively for $\log n$ levels leads to the superpolynomial speedup for quantum algorithms.

5 Recursive Fourier Sampling

Our goal is to give a superpolynomial separation between quantum computation and classical probabilistic computation. The idea is to define a recursive version of the Fourier sampling problem, where each query to the function (on an input of length n) is itself the answer to a recursive Fourier sampling problem (on an input of length n/2). Intuitively a classical algorithm would need to solve n subproblems to solve a problem on an input of length n (since it must make n queries). Thus its running time satisfies the recurrence $T(n) \geq nT(n/2) + O(n)$ which has solution $T(n) = \Omega(n^{\log n})$. The quantum algorithm needs to make only two queries and thus its running time satisfies the recurrence $T(n) = 2T(n/2) + O(n)$, which solves to $T(n) = O(n \log n)$.

Recall that for one level we have an oracle for $f(x)$ with the promise that $f(x) = s \cdot x$. For two levels, we are given functions $f(x)$ and $f'(x,y)$ with the promise that for some $g(x)$ we have $f(x) = s \cdot g(x)$ and

$f'(x,y) = g(x) \cdot y$. To compute s we must know a few values of $g(x)$, and to compute each of these values we sample y and use f'. More generally, we are given oracles to $f_1, \ldots, f_k$ and g_k with the promise that $f_i(x_1, \ldots, x_i) = g_{i-1}(x_1, \ldots, x_{i-1}) \cdot g_i(x_1, \ldots, x_i)$ and we wish to compute $g_0 \equiv s$. Illustrated below is level three:

1. To find s, sample enough values from $g_1(\cdot)$ and use the promise $f_1(x_1) = s \cdot g_1(x_1)$.
2. To sample $g_1(x_1)$ for some particular x_1, sample enough values from $g_2(x_1, \cdot)$ and use the promise $f_2(x_1,x_2) = g_1(x_1) \cdot g_2(x_1,x_2)$.
3. To sample $g_2(x_1,x_2)$ for some particular x_1,x_2, use the oracle for $g_3(x_1,x_2,\cdot)$ and the promise $f_3(x_1,x_2,x_3) = g_2(x_1,x_2) \cdot g_3(x_1,x_2,x_3)$.

It should be clear now why this is the procedure we want. For the lengths to scale appropriately, we must have $2|x_i| = |x_{i-1}|$ and $k = \log n$.

The proof that no classical probabilistic algorithm can reconstruct s is somewhat technical, and establishes that for a random g satisfying the promise, any algorithm (deterministic or probabilistic) that makes $n^{o(\log n)}$ queries to g must give the wrong answer on at least $1/2 - o(1)$ fraction of g's. This lemma continues to hold even if the actual queries are chosen by a helpful(but untrusted) genie who knows the answer.

This establishes that relative to an oracle $BQP \not\subseteq MA$. MA is the probabilistic generalization of NP. It is conjectured that recursive Fourier sampling does not lie in the polynomial hierarchy. In particular, it is an open question to show that, relative to an oracle, recusive Fourier sampling does not lie AM or in BPP^{NP}. The latter class is particularly important since it includes approximate counting.

6 $BQP \subseteq PSPACE$

To put this gap result in context, we give an important complexity result relating BQP to $PSPACE$.

Theorem 3.1: $P \subseteq BPP \subseteq BQP \subseteq P^{\#P} \subseteq PSPACE$.

We give a sketch of the proof that $BQP \subseteq P^{\#P}$. We assume without loss of generality that all the transition amplitudes specified in the transition function δ are real (exercise). The action of a quantum circuit may be described by a tree, each node is labelled with a computational basis state, i.e. a bit string. The root of the tree corresponds to the input $|x\rangle$ and applying a gate to any node yields a superposition of basis states represented by the children of that node. We label the edge to each child by the corresponding amplitude. Let us assume that the quantum circuit accepts or rejects depending upon whether the first qubit, when measured in the computational basis is 0 or 1. Thus each leaf of the tree is either an accepting or rejecting node depending on whether the first bit of the string labeling it is 0 or 1. The amplitude of a path p from the root to a leaf of the tree, β_p, is just the product of the branching amplitudes along the path, and is computable to within $1/2^j$ in time polynomial in j. Several paths may lead to the same configuration c. Thus the amplitude of c after application of T gates is the following sum over all T length paths p: $\alpha_c = \sum_{p\ to\ c} \beta_p$. The probability that quantum circuit accepts is $\sum_{accepting\ c} |\alpha_c|^2$. Let $a_p = max(\beta_p, 0)$ and $b_p = max(-\beta_p, 0)$. Then $|\alpha_c|^2$ can be written as $|\alpha_c|^2 = \sum_{p\ to\ c}(a_p - b_p)^2 = \sum_{p\ to\ c} a_p^2 + b_p^2 - \sum_{p,p'\ to\ c} 2a_p b_p$. It follows that the acceptance probability of the quantum circuit can be written as the difference between the two quantities $\sum_{accepting\ c} \sum_{p\ to\ c} a_p^2 + b_p^2$, and $\sum_{accepting\ c} \sum_{p,p'\ to\ c} 2a_p b_{p'}$. Since each of these quantities is easily seen to be in $P^{\#P}$, it follows that $BQP \subseteq P^{\#P}$.

In view of this theorem, we cannot expect to prove that BQP strictly contains BPP without resolving the long standing open question in computational complexity theory, namely, whether or not $P = PSPACE$.

6.1 Extended Church-Turing Thesis

The extended Church-Turing thesis is a foundational principle in computer science. It asserts that any "reasonable" model of computation can be efficiently simulated on a standard model such as a Turing Machine or a Random Access Machine or a cellular automaton. This thesis forms the foundation of complexity theory — for example ensuring that that the class P (polynomial time) is well defined. But what do we mean by "reasonable"? In this context, reasonable means "physically realizable in principle". One constraint that this places is that the model of computation must be digital. Thus analog computers are not reasonable models of computation, since they assume infinite precision arithmetic. In fact, it can be shown that with suitable infinite precision operations, an analog computer can solve NP-Complete problems in polynomial time. And an infinite precision calculator with operations +, x, =0?, can factor numbers in polynomial time.

We first establish that quantum computers are digital:

7 Is Quantum Computation Digital?

There is an issue as to whether or not quantum computing is digital. We need only look at simple gates such as the Hadamard gate or a rotation gate to find real values.

$$H = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \qquad R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \tag{1}$$

When we implement a gate, how accurate does it need to be? Do we need infinite precision to build this gate properly? A paper by Shamir, "How To Factor On Your Calculator," shows that if we assume infinite precision arithmetic, then some NP complete problems can be solved in polynomial time. However, we obviously cannot have infinite precision, so we must digitize quantum computation in order to approximate values such as $1/\sqrt{2}$. It turns out that $\log n$ bits of precision are necessary.

Suppose we want to build a gate that rotates the input by θ, but the best accuracy we can actually build is rotation by $\theta \pm \Delta\theta$ (finite precision). Let $U_1, \ldots, U_m$ be a set of ideal gates that implement an exact rotation by θ. Let $V_1, \ldots, V_m$ be a set of actual (constructible) gates that implement rotation by $\theta \pm \Delta\theta$. Let $|\phi\rangle$ be the initial state. Let $|\psi\rangle$ be the ideal output

$$|\psi\rangle = U_1 U_2 \cdots U_m |\phi\rangle, \tag{2}$$

and let $|\psi'\rangle$ be the actual output

$$|\psi'\rangle = V_1 V_2 \cdots V_m |\phi\rangle. \tag{3}$$

The closer $|\psi\rangle$ and $|\psi'\rangle$ are to each other, the better the approximation. If we can approximate each gate to within $\varepsilon = O(1/m)$, then we can approximate the entire circuit with small constant error.

Theorem 3.2: *If* $\|U_i - V_i\| \leq \frac{\varepsilon}{4m}$ *for* $1 \leq i \leq m$, *then* $\| |\psi\rangle - |\psi'\rangle \| \leq \frac{\varepsilon}{4}$.

Proof:Consider the two hybrid states

$$\begin{aligned} |\psi_k\rangle &= U_1 \cdots U_{k-1} V_k \cdots V_m |\phi\rangle \quad \text{, and} \\ |\psi_{k+1}\rangle &= U_1 \cdots U_k V_{k+1} \cdots V_m |\phi\rangle. \end{aligned}$$

Subtract ψ_{k+1} from ψ_k to get

$$|\psi_k\rangle - |\psi_{k+1}\rangle = U_1 \cdots U_{k-1}(V_k - U_k) V_{k+1} \cdots V_m |\phi\rangle \tag{4}$$

Since the unitary transformations don't change the norm of the vector, the only term we need to consider is $U_{k+1} - V_{k+1}$. But we have an upper bound on this, so we can conclude that

$$\| |\psi_k\rangle - |\psi_{k+1}\rangle \| \leq \frac{\varepsilon}{4m}. \tag{5}$$

Another way to see this is that applying unitary transformations to $U_m|\phi\rangle$ and $V_m|\phi\rangle$ preserves the angle between them, which is defined to be the norm.

We use the triangle inequality to finish to proof.

$$\begin{aligned} \| |\psi\rangle - |\psi'\rangle \| &= \| |\psi_0\rangle - |\psi_m\rangle \| \\ &\leq \sum_{i=0}^{m-1} \| |\psi_i\rangle - |\psi_{i+1}\rangle \| \\ &\leq m \cdot \frac{\varepsilon}{4m} \leq \frac{\varepsilon}{4}. \end{aligned}$$

We have already seen that quantum computers are digital computers, and therefore a reasonable model of computing. But we also established that $P \subseteq BPP \subseteq BQP \subseteq PSPACE$. Since we do not know how to separate P from $PSPACE$, it follows that we cannot unconditionally prove that quantum computers are more powerful than classical computers. Instead there are two ways of establishing that quantum computers are more powerful than classical computers: by an oracle separation, or by giving an efficient quantum algorithm for a problem believed to be hard for classical algorithms. Historically, the first demonstrations that quantum computers are more powerful than classical computers were by proving oracle separations, starting with the recursive Fourier sampling problem, which we will outline below. We will briefly sketch this below and discuss the conjecture that recursive Fourier sampling does not lie in the polynomial hierarchy. The next oracle separation, Simon's problem, provided the basic template that Shor followed in his quantum algorithm for factoring.

8 Quantum Circuit Implementation

As an aside, let us think about how one might implement a quantum computer. One way to do it would be to have an environment state with n photons. Then for a bit flip operation, the qubit either absorbs an electron or emits one. This scheme unfortunately entangles the qubit with the environment however:

$$(|0\rangle + |1\rangle) \otimes |n\rangle_e \rightarrow |1\rangle |n-1\rangle_e + |0\rangle |n+1\rangle_e \tag{6}$$

This seems like an unsurmountable problem, but one can imagine that the environment state would have many more electrons than there are qubits. Slightly more precisely, if the environment is in state $|\phi\rangle$ then after a bit flip it would be in state $|\phi'\rangle$, where $\phi \cong \phi'$. In fact, one can show that $|\phi - \phi'| \cong 1/\sqrt{n}$ as $n \rightarrow \infty$.

We will think about our quantum computations as an array of n qubits and a classical computer as a controller, which chooses qubits and performs 'gates' on them sequentially. It is still not clear though that quantum mechanical theory will hold when there are many qubits.

9 Communication Complexity of Inner Product Function

Suppose Alice has x and Bob has y. Show that it requires at least $\Omega(n)$ communications between Alice and Bob to compute $x \cdot y$. (Hint: use the classical algorithm and the Hadamard gate.)

10 Simon's Algorithm

Recall that our basic primitive for designing quantum algorithms is Fourier sampling: prepare some quantum state $|\psi\rangle = \sum_x \alpha_x |x\rangle$ on n qubits; perform a Hadamard transform, resulting in the superposition $\sum_x \beta_x |x\rangle$; now measure to sample x with probability $|\beta_x|^2$. The point is that classically it is difficult to simulate the effects of the quantum interference, and therefore to determine for which strings x there is constructive interference and are therefore output with high probability.

We now consider a new way of setting up the initial superposition $|\psi\rangle = \sum_x \alpha_x |x\rangle$.

10.1 Setting up a random pre-image state

Suppose we're given a classical circuit for a $k-1$ function $f : \{0,1\}^n \to \{0,1\}^n$.

We will show how to set up the quantum state $|\phi\rangle = 1/\sqrt{k}\sum_{x:f(x)=a} |x\rangle$. Here a is uniformly random among all a in the image of f.

The algorithm uses two registers, both with n qubits. The registers are initialized to the basis state $|0\cdots 0\rangle |0\cdots 0\rangle$. We then perform the Hadamard transform $H^{\otimes n}$ on the first register, producing the superposition

$$\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle |0\cdots 0\rangle .$$

Then, we compute $f(x)$ through the oracle C_f and store the result in the second register, obtaining the state

$$\frac{1}{2^{n/2}} \sum_{x\in\{0,1\}^n} |x\rangle |f(x)\rangle .$$

The second register is not modified after this step. Thus we may invoke the principle of safe storage and assume that the second register is measured at this point.

Let a be the result of measuring of the second register. Then a is a random element in the range of f, and according to rules of partial measurement, the state of the first register is a superposition over exactly those values of x that are consistent with those contents for the second register. i.e.

$$|\phi\rangle = 1/\sqrt{k} \sum_{x:f(x)=a} |x\rangle$$

10.2 The Algorithm

Suppose we are given function $2-1$ $f : \{0,1\}^n \to \{0,1\}^n$, specified by a black box, with the promise that there is an $a \in \{0,1\}^n$ with $a \neq 0^n$ such that

- For all x $f(x+a) = f(x)$.
- If $f(x) = f(y)$ then either $x = y$ or $y = x+a$.

The challenge is to determine a. It is intuitively obvious that this is a difficult task for a classical probabilistic computer. We will show an efficient quantum algorithm.

1. Use f to set up random pre-image state

$$\phi = 1/\sqrt{2}|z\rangle + 1/\sqrt{2}|z+a\rangle$$

$|0^n\rangle H_{2^n} C_f\ H_{2^n} |y\rangle$
$|0^n\rangle \qquad |f(x)\rangle$

Figure 1: Simon's algorithm

where z is a random n-bit string.

2. Perform a Hadamard transform $H^{\otimes n}$.

After step 2 we obtain a superposition

$$\sum_{y \in \{0,1\}^n} \alpha_y |y\rangle$$

where

$$\alpha_y = \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot z} + \frac{1}{\sqrt{2}} \frac{1}{2^{n/2}} (-1)^{y \cdot (z \oplus a)} = \frac{1}{2^{(n+1)/2}} (-1)^{y \cdot z} \left[1 + (-1)^{y \cdot a}\right].$$

There are now two cases. For each y, if $y \cdot a = 1$, then $\alpha_y = 0$, whereas if $y \cdot a = 0$, then

$$\alpha_y = \frac{\pm 1}{2^{(n-1)/2}}.$$

So when we observe the first register, with certainty we'll see a y such that $y \cdot a = 0$. Hence, the output of the measurement is a random y such that $y \cdot a = 0$. Furthermore, each y such that $y \cdot a = 0$ has an equal probability of occurring. Therefore what we've managed to learn is an equation

$$y_1 a_1 \oplus \cdots \oplus y_n a_n = 0 \tag{7}$$

where $y = (y_1, \ldots, y_n)$ is chosen uniformly at random from $\{0,1\}^n$. Now, that isn't enough information to determine a, but assuming that $y \neq 0$, it reduces the number of possibilities for a by half.

It should now be clear how to proceed. We run the algorithm over and over, accumulating more and more equations of the form in (7). Then, once we have enough of these equations, we solve them using Gaussian elimination to obtain a unique value of a. But how many equations is enough? From linear algebra, we know that a is uniquely determined once we have $n-1$ linearly independent equations—in other words, $n-1$ equations

$$\begin{aligned} y^{(1)} \cdot a &\equiv 0 \pmod 2 \\ &\vdots \\ y^{(n-1)} \cdot a &\equiv 0 \pmod 2 \end{aligned}$$

such that the set $\left\{y^{(1)}, \ldots, y^{(n-1)}\right\}$ is linearly independent in the vector space Z_2^n. Thus, our strategy will be to lower-bound the probability that any $n-1$ equations returned by the algorithm are independent.

Suppose we already have k linearly independent equations, with associated vectors $y^{(1)}, \ldots, y^{(k)}$. The vectors then span a subspace $S \subseteq Z_2^n$ of size 2^k, consisting of all vectors of the form

$$b_1 y^{(1)} + \cdots + b_k y^{(k)}$$

with $b_1, \ldots, b_k \in \{0,1\}$. Now suppose we learn a new equation with associated vector $y^{(k+1)}$. This equation will be independent of all the previous equations provided that $y^{(k+1)}$ lies *outside* of S, which in turn has probability at least $(2^n - 2^k)/2^n = 1 - 2^{k-n}$ of occurring. So the probability that any n equations are independent is exactly the product of those probabilities.

$$\left(1-\frac{1}{2^n}\right)\times\left(1-\frac{1}{2^{n-1}}\right)\times\cdots\times\left(1-\frac{1}{4}\right)\times\left(1-\frac{1}{2}\right).$$

Can we lower-bound this expression? Trivially, it's at least

$$\prod_{k=1}^{\infty}\left(1-\frac{1}{2^k}\right)\approx 0.28879;$$

the infinite product here is related to something in analysis called a q-series. Another way to look at the constant $0.28879\ldots$ is this: it is the limit, as n goes to infinity, of the probability that an $n \times n$ random matrix over Z_2 is invertible.

But we don't need heavy-duty analysis to show that the product has a constant lower bound. We use the inequality $(1-a)(1-b) = 1-a-b+ab > 1-(a+b)$, if $a,b \in (0,1)$. We just need to multiply the product out, ignore monomials involving two or more $\frac{1}{2^k}$ terms multiplied together (which only increase the product), and observe that the product is lower-bounded by

$$\left[1-\left(\frac{1}{2^n}+\frac{1}{2^{n-1}}+\cdots+\frac{1}{4}\right)\right]\cdot\frac{1}{2}\geq\frac{1}{4}.$$

We conclude that we can determine a with constant probability of error after repeating the algorithm $O(n)$ times. So the number of queries to f used by Simon's algorithm is $O(n)$. The number of computation steps, though, is at least the number of steps needed to solve a system of linear equations, and the best known upper bound for this is $O\left(n^{2.376}\right)$, due to Coppersmith and Winograd.

10.3 Classical solution

We are going to prove that any probabilistic algorithm needs an exponential time to solve this problem. Suppose that a is chosen uniformly at random from $\{0,1\}^n - \{0^n\}$. Now consider a classical probabilistic algorithm that's already made k queries, to inputs $x_1,\ldots,x_k$. We want to know how much information the algorithm could have obtained about a, given those queried pairs $(x_i, f(x_i))$.

On the one hand, there might be a pair of inputs x_i, x_j (with $1 \leq i,j \leq k$) such that $f(x_i) = f(x_j)$. In this case, the algorithm already has enough information to determine a: $a = x_i \oplus x_j$.

On the other hand, suppose no such pair $f(x_i), f(x_j)$ exists. Then the queried $f(x_i)$'s are distinct and a is none of $\binom{k}{2}$ values $x_i \oplus x_j$.

The probability that the next query will succeed is at most

$$\frac{k}{2^n - 1 - \binom{k}{2}}$$

because there are at least $2^n - 1 - \binom{k}{2}$ possible values of u for choosing at the $(k+1)$-th query. And $f(x_{k+1})$ should be equal to one of the prior observed $f(x_i)$, $i \in [1,k]$.

Taking the sum over all $k \in \{1, \ldots, m\}$. We get

$$\sum_{k=1}^{m} \frac{k}{2^n - 1 - \binom{k}{2}} \leq \sum_{k=1}^{m} \frac{k}{2^n - k^2} \leq \frac{m^2}{2^n - m^2}$$

In order to have an constant probability, we must choose $m = \Omega(2^{n/2})$. Hence, any deterministic algorithm has to run in exponential time to get a correct answer with probability larger than a constant.