1. a) Show that $\oplus$ (Parity) is $\text{AC}^0$-reducible to binary multiplication. Here binary multiplication is the function $\text{MULT} : \{0, 1\}^n \times \{0, 1\}^n \rightarrow \{0, 1\}^{2n}$ is defined by $\text{MULT}(x, y) = z$ if $z$ if $a \cdot b = c$ where strings $x, y, z$ represent natural numbers $a, b, c$ in binary.

To show the $\text{AC}^0$ reduction, define an $\text{AC}^0$ function $f$ such that $f(x) = (y, z, i)$, where the string $x$ has an odd number of 1’s iff the $i$th bit of $\text{MULT}(y, z)$ is 1. Do this as follows: Specify a number $k$, and let $y$ be $x$ with each of its consecutive bits separated by $k$ 0’s, and let $z$ be constructed similarly. (The idea is to remove the carries that would arise from normal long-hand multiplication of $x$ times $y$.) Show that this is an $\text{AC}^0$ reduction.

Solution:
Following the Hint, we wish to define an $\text{AC}^0$ function $f(x) = (y, z, i)$, where the string $x$ has an odd number of 1’s iff the $i$th bit of $\text{MULT}(y, z)$ is 1. We will specify $k$ later.

Suppose $x$ in binary is the string $x_n x_{n-1} \cdots x_0$, so $x = \sum_{i=0}^{n} x_i 2^i$. Then we set $y$ to be $x$ with consecutive bits separated by $k$ 1s, and define $z$ to be what $y$ would be if all the bits of $x$ are 1. Then

$$y = x_n 0^k x_{n-1} 0^k \cdots 0^k x_0 = \sum_{i=0}^{n} x_i 2^{(k+1)i}$$

and

$$z = 10^k 10^k \cdots 0^k 1 = \sum_{j=0}^{n} 2^{(k+1)j}$$

Then

$$y \cdot z = \left( \sum_{i=0}^{n} x_i 2^{(k+1)i} \right) \left( \sum_{j=0}^{n} 2^{(k+1)j} \right)$$

$$= \sum_{\ell=0}^{2n} \left( \sum_{i+j=\ell} x_i \right) 2^{(k+1)\ell}$$

Then setting $w = y \cdot z$, the contribution to $w$ in the above sum when $\ell = n$ is

$$2^{(k+1)n} \sum_{i=0}^{n} x_i$$

Now we choose $k$ to be larger than $\log n$ (say $k = n$), then the contribution for $\ell < n$ in the above sum does not affect the bit number $(k+1)n$ in the binary notation for $w$. Hence this bit is precisely $\oplus(x_n, \ldots, x_0)$.

Hence we define $f(x) = (y, z, (k+1)n)$. This is trivially an $\text{AC}^0$ function of the bits of the binary notation of $x$. In fact, for fixed $n$, each bit of the $f(x)$ is either copied from
a bit of \( x \), or it does not depend on the input \( x \) at all, so it is hard wired to be 0 or 1. So the \( \text{AC}^0 \) circuit needs no gates.

b) Conclude that \text{MULT} is not an \( \text{AC}^0 \) function.

\textbf{Solution:}
If \text{MULT} were an \( \text{AC}^0 \) function, then this \( \text{AC}^0 \) circuit applied to the output of \( f(x) \) above would provide an \( \text{AC}^0 \) circuit computing parity. But no such \( \text{AC}^0 \) circuit exists.

2. The point of this exercise is to show that Theorem 14.4, page 291 (Razborov-Smolensky) generalizes to the case that \( p \) and \( q \) are positive powers of distinct primes. (We use \( \text{AC}^0(m) \) for \( \text{ACC}_0(m) \).)

(a) Show that if \( a, b > 1 \) then \( \text{MOD}_a \) is in \( \text{AC}^0(ab) \).

\textbf{Solution:}
Recall that a \( \text{MOD}_m \) gate outputs 0 iff the total number of 1 bits among its inputs is divisible by \( m \).

Feed \( b \) copies of the input to a \( \text{MOD}_{ab} \) gate \( g \). Then the output of \( g \) is 0 iff the total number of ones in the input is divisible by \( a \). This circuit has depth 1 and is size linear in the size of the input, so it is clearly an \( \text{AC}^0 \) circuit.

(b) Show that if \( a, b > 1 \) then \( \text{MOD}_{ab} \) is in \( \text{AC}^0(a, b) \).

\textbf{Solution:}
Observe that if \( x \) is the number input bits that are 1, then \( x \mod ab = 0 \) iff \( x \mod a = 0 \) and \( x/a \mod b = 0 \). We can implement this in an \( \text{AC}^0(a, b) \) circuit as follows.

Feed all input bits into a \( \text{MOD}_a \) gate, and feed the output of this into an OR gate, which will be the output gate \( \text{OUT} \). (Note that this OR gate acts like an AND gate when we interpret 0 as TRUE and 1 as FALSE.) The other input to \( \text{OUT} \) is the output of a \( \text{MOD}_b \) gate whose input is a sequence of wires whose total number of ones is \( x/a \) (assuming \( x \) is divisible by \( a \)).

In more detail for the other input: For each \( i \), feed the first \( i \) inputs into a \( \text{MOD}_a \) gate \( g_i \). Feed \( \neg g_i \) and the \( i \)th input bit \( x_i \) into an AND gate \( h_i \). Thus if \( x = ac \), then \( h_i \) has output 1 iff \( x_i \) causes the sum of the first \( i \) input bits to reach \( ac' \) for some integer \( c' \leq c \).

Now feed the output of each \( h_i \) into one \( \text{MOD}_b \) gate, and let the output of the \( \text{MOD}_b \) gate be the other input to \( \text{OUT} \). The circuit depth is 5, and the circuit has \( O(n^2) \) gates, so it is an \( \text{AC}^0 \) circuit.

(c) Conclude from the above that if \( i, j > 0 \) and \( m > 1 \), then \( \text{MOD}_{m^i} \) is in \( \text{AC}^0(m^j) \).

\textbf{Solution:}
By Part a), \( \text{MOD}_m \) is in \( \text{AC}^0(m^j) \). Hence by repeated applications of Part b), we see successively that each of \( \text{MOD}_m, \text{MOD}_{m^2}, \ldots \text{MOD}_{m^i} \) is in \( \text{AC}^0(m^j) \).

3. Let \#MATCHINGS(\( G \)) be the number of matchings of a bipartite graph \( G \). (A matching is any set of edges of \( G \) such that no two edges in the set share a common vertex.)
Let \( \#\text{MON2SAT}(\varphi) \) be the number of satisfying assignments of a monotone 2CNF formula \( \varphi \).

Use the fact that \( \#\text{MATCHINGS}(G) \) is \#P complete to show that \( \#\text{MON2SAT}(\varphi) \) is \#P complete.

**Solution:**
Since there is a polytime algorithm to check whether a truth assignment satisfies a 2CNF formula, it is clear that \( \#\text{MON2SAT} \) is in \#P. So it suffices to show that \( \#\text{MATCHINGS} \) is polynomial time reducible to \( \#\text{MON2SAT} \).

Given a bipartite graph \( G = (V,W,E) \), we introduce a Boolean variable \( x_e \) for each edge \( e \in E \), and let \( \varphi_G \) be the conjunction of all clauses of the form \((x_e \lor x_{e'})\), where the distinct edges \( e \) and \( e' \) share a common vertex. If we interpret \( x_e \) as meaning that edge \( e \) is missing from the matching, then there is a one-one correspondence between matchings in \( G \) and truth assignments to the variables \( \{x_e \mid e \in E\} \) which satisfy \( \varphi \).

However it is possible that some variables \( x_e \) do not occur in \( \varphi \). Specifically if \( I \) is the set of ‘isolated’ edges in \( E \) (i.e. edges that do not have a vertex in common with any other edge), then for \( e \in E \), the variable \( x_e \) does not occur in \( \varphi \) iff \( e \in I \). If we define

\[
m = |I|
\]

then \( m \) is the number of variables \( x_e \) which do not occur in \( \varphi \). Note that any of the \( 2^m \) subsets of \( I \) can occur in a matching of \( G \) without affecting the other edges in the matching.

Thus the polytime algorithm reducing \#MATCHINGS to \#MON2SAT is the following: Given \( G \), compute \( \varphi_G \), ask the \#MON2SAT oracle to return the number \( N \) of satisfying assignments to \( \varphi_G \), compute \( m = |I| \), and output \( 2^m N \).

4. Show that \( \text{P}^{\text{PP}} = \text{P}^{\text{#P}} \). (See page 345 for the definition of PP.)

**Solution:**
Obviously determining whether a majority of strings \( y \in \{0,1\}^{p(|x|)} \) satisfy a polytime relation \( R(x,y) \) is polytime reducible to counting the number of such strings \( y \) which satisfy \( R(x,y) \), and hence \( \text{P}^{\text{PP}} \subseteq \text{P}^{\text{#P}} \).

The reverse inclusion is based on the proof of Lemma 17.7 in the text (see below).

**NOTE:** Another approach is show how to reduce \#SAT to \#PSAT. Part of this reduction involves describing a polytime function which takes a binary number \( M > 0 \) with \( n \) bits to a Boolean formula \( \varphi \) which is satisfied by exactly \( M \) truth assignments to its variables. Rather than give an obscure recursive construction for \( \varphi \) (done by several students) it is better to describe \( \varphi \) explicitly as follows:

Let \( M \) be the bit string \( m_1m_2\cdots m_n \). Then \( \varphi \) has \( n \) variables \( x_1, \ldots, x_n \) giving the bits of the binary notation for an arbitrary natural number \( N < M \). Thus \( \varphi \) is \((\varphi_1 \lor \cdots \lor \varphi_n)\) where \( \varphi_j \) is \((x_j \land \neg x_j) \) if \( m_j = 0 \) and if \( m_j = 1 \) then \( \varphi_j \) is \((\ell_1 \land \cdots \land \ell_{j-1} \land \neg x_j)\), where \( \ell_i = x_i \) if \( m_i = 1 \) and \( \ell_i = \neg x_i \) if \( m_i = 0 \).
Now we show how to adapt the proof of Lemma 17.7. Let $f$ be a function in #P. It suffices to show that $f$ is polytime reducible to PP. By definition of #P, there is a polytime relation $R(x, y)$ such that $f(x) = \#^m_R(x)$, where we use the notation

$$\#^m_R(x) = |\{y \in \{0, 1\}^m : R(x, y)\}|$$

where $m = p(|x|)$ for some polynomial $p(n)$.

For every two relations $R_0(x, y)$ and $R_1(x, y)$ we denote by $R_0 + R_1$ the relation $R'$ given by, for $b \in \{0, 1\}$

$$R'(x, by) \iff R_b(x, y)$$

Thus

$$\#^{m+1}_{R_0 + R_1}(x) = \#^m_{R_0}(x) + \#^m_{R_1}(x)$$

Also for $N \in \{0, \ldots, 2^m\}$ we define $R_N$ by

$$R_N(x, y) \iff y < N$$

where we treat $y$ as a binary number. Thus

$$\#^m_{R_N}(x) = N$$

A polytime TM with oracle access to PP can determine whether

$$\#^m_{R_N + R}(x) = N + \#^m_R(x) \geq 2^m$$

(1)

To compute $\#^m_R(x)$ we can use binary search to find the smallest $N$ that satisfies (1). For this $N$, (1) becomes an equality, so $\#^m_R(x) = 2^m - N$. 

4