Numbers of theorems, lemmas, definitions, examples, sections, and pages are from Arora and Barak’s book.

Answer 1
Recall that the permanent of an $n$-by-$n$, integer-valued matrix $A$ is defined by the formula

$$\text{Perm}(A) = \sum_{\sigma \in S_n} \prod_{i=1}^{n} A_{i \sigma(i)}$$  \hspace{1cm} (*) \hspace{1cm} (a)

A general $n$-by-$n$, integer-valued matrix $A$ corresponds to a weighted digraph $G_A$ as follows. The vertices of $G_A$ are $\{v_1, \ldots, v_n\}$. If $A_{ij} = 0$, then there is no directed edge $(v_i, v_j)$ in $G_A$. If $A_{ij} = k \neq 0$, then there is a directed edge $(v_i, v_j)$ in $G_A$ of weight $k$. Self-loops are allowed, because the diagonal entries of $A$ may be nonzero. A cycle cover of a digraph is a set $\{C_1, \ldots, C_m\}$ of cycles that “covers” $G_A$ in the following sense: In the sub-digraph consisting of vertex set $\{v_1, \ldots, v_n\}$ and exactly those edges in $C_1, \ldots, C_m$, each vertex has in-degree 1 and out-degree 1. For general integer matrices, $\text{Perm}(A)$ is defined to be the sum, over all cycle covers $\{C_1, \ldots, C_m\}$, of the product of the weights on all the directed edges in $\{C_1, \ldots, C_m\}$. To see why this is equivalent to (*), note that there is a one-to-one correspondence between permutations $\sigma \in S_n$ and potential cycle covers. Basic group theory tells us that each $\sigma \in S_n$ has a unique decomposition into disjoint cycles $(i_1 \ldots i_j)(i_{j+1} \ldots i_k) \ldots$; this notation is interpreted to mean that $\sigma$ maps $i_1$ to $i_2$, $i_2$ to $i_3$, $i_3$ to $i_1$, $i_{j+1}$ to $i_{j+2}$, $i_{j+2}$ to $i_{j+1}$, etc.; cycles of size one are allowed, because $\sigma$ may have fixed points. This cycle decomposition of $\sigma$ corresponds to a cycle cover of $G_A$ if and only if each pair of consecutive indices $i, \sigma(i)$ in one of the cycles of the decomposition corresponds to a directed edge $(v_i, v_{\sigma(i)})$ in $G_A$; furthermore, these are the only $\sigma$’s for which the product of the $A_{i \sigma(i)}$ is nonzero.

(b) This is a special case of Example 17.12.

Answer 2
(a) A $k$-round, public-coin interactive proof system is a $k$-round interactive proof system in which every message that the verifier sends to the prover is a string chosen uniformly at random from $\{0, 1\}^m$, where $m$ is a polynomially bounded function of the input length $n$. Note that the prover does not know which random string the verifier will send in round $i$ when he sends his response in round $i-1$.

(b) Section 8.3.3 gives an interactive proof system $(P, V)$ for the PSPACE-complete language TQBF. Because IP=PSPACE and TQBF is PSPACE-complete, one can use $(P, V)$ for any language $L$ that has an interactive proof system: Simply reduce an instance $x$ of membership in $L$ to a quantified boolean formula $\varphi$; then $x$ is in $L$ if and only if $(P, V)$ can be used to prove that $\varphi$ is true. $(P, V)$ is in fact a public-coin interactive proof system; therefore, restriction to public coins does not reduce the power of interactive proof systems. (That is, the restriction does not reduce the language-recognition power of interactive proof systems; for some $L$, it may cause an increase in the required number of rounds.)
(c) See Definition 8.26. It is similar to Definition 8.6 in that the “program” plays the role of a prover, but there are three key differences. Focus attention on the case in which the “computational task” is to decide membership in the language L. A checker should work for both yes instances and no instances; by contrast, one could have different interactive proof systems for L and its complement (think graph isomorphism). In the case of checking, a correct prover or a cheating prover must be a program (or “oracle”) that is fixed in advance of the computation; by contrast, the prover in Definition 8.6 could strategize during the computation. (This is the difference that we discussed when contrasting one-prover and multi-prover interactive proof systems.) Finally, in the case of checking, the correct prover (or program) is limited to the amount of computational power required to decide L; by contrast, in Definition 8.6, there is no upper bound on the computational power of the prover. (In fact, there are languages L in IP for which no known proof system has a prover in FP^c.)

Answer 3
(a) See the Goldwasser-Sipser lower-bound protocol on page 137.
(b) Because BPP is closed under complement, it suffices to show that it is in \( \sum_p^P \)
For every L in BPP, there is a machine M with the properties given in the question.
Consider the formula \( \exists u_1, \ldots, u_k \forall r M(x, r \oplus u_1) \lor \cdots \lor M(x, r \oplus u_k) \).
Here x is a string in \( \{0, 1\}^n \) for which we want to determine membership in L, \( u_i \) and \( r \) are strings in \( \{0, 1\}^m \), where m is a polynomially bounded function of n, and \( k = 1 + \frac{m}{n} \).
If x is in L, there is a \( U = \{u_1, \ldots, u_k\} \) such that all r are in the neighborhood of \( S_x \) in \( G_U \), where \( S_x \) is the (big) set of strings s in \( \{0, 1\}^m \) for which \( M(x, s) = 1 \); thus, for all r there is at least one \( u_i \) such that \( r \oplus u_i \) is in \( S_x \), which implies that \( M(x, r \oplus u_i) = 1 \). On the other hand, if x is not in L, then \( S_x \) is small, and so all \( U \) are such that there is at least one r not in the neighborhood of \( S_x \) in \( G_U \).

Answer 4
(a) True (Lemma 17.17)
(b) True (Toda’s Theorem [Thm 17.14])
(c) Unknown
(d) False. Although BPP \( \subseteq P/poly \), the inclusion is proper, because there are undecidable sets in \( P/poly \) but not in BPP.

Answer 5
(a) The negative result about approximation within a factor of 2 is a consequence of the following version of the fact that every language L in NEXP has a ppt “oracle” proof system: There is a ppt Turing Machine M such that, for every x in L, there is an oracle O such that \( M^O \) accepts x with probability 1, and, for every x not in L, for all oracles \( O^* \), \( M^{O^*} \) accepts x with probability less than 1/4. From standard Chernoff-bound-based techniques for reducing the error probability of a ppt computation, we know that, for any constant c, there is also a ppt Turing Machine T such that, for every x in L, there is an oracle O such
that \( T^O \) accepts \( x \) with probability 1, and, for every \( x \) not in \( L \), for all oracles \( O^* \), \( T^{O^*} \) accepts \( x \) with probability less than \( c/2 \). Using \( T \) in the proof of the clique-nonapproximability theorem yields a negative result about approximation within a factor of \( c \).

(b) See definitions 7.2, 7.3, and 7.7. It is immediate from the definitions of \( \text{RP} \), \( \text{coRP} \), and \( \text{BPP} \) that both \( \text{RP} \) and \( \text{coRP} \) are subsets of \( \text{BPP} \). Therefore, Theorem 7.8 implies that \( \text{ZPP} \) is also a subset of \( \text{BPP} \). Stating Theorem 7.8 without proof sufficed for full credit on this question, but the proof is included here for your information.

Let \( L \) be a language in \( \text{RP} \cap \text{coRP} \), where \( M_1 \) is a ppt Turing Machine that accepts \( L \) and \( M_2 \) is a ppt Turing Machine that accepts the complement of \( L \). Let \( q(n) \) be the larger of the running times of \( M_1 \) and \( M_2 \). The following ppt Turing Machine \( M \) clearly accepts \( L \): On input \( x \) of length \( n \), run each of \( M_1 \) and \( M_2 \) \( s(n) \) times on input \( x \), using independent coin tosses for each run. If on any of these runs \( M_1 \) accepts \( x \), then \( M \) accepts \( x \) and halts. Similarly, if on any of these runs \( M_2 \) accepts \( x \), then \( M \) rejects \( x \) and halts. If \( M \) finishes these \( s(n) \) independent runs of \( M_1 \) and \( M_2 \) without halting, then it simulates \( M_1 \) on all possible coin-toss sequences (accepting \( x \) and halting if \( M_1 \) ever accepts) and then simulates \( M_2 \) on all possible coin-toss sequences (rejecting \( x \) and halting if \( M_2 \) ever accepts). Note that \( M \) always outputs the correct answer, as required by the definition of \( \text{ZPP} \). Suppose that the probability that \( M \) finishes the \( s(n) \) independent runs of \( M_1 \) and \( M_2 \) without halting is at most \( 2^{-q(n)} \). Then the expected running time of \( M \) on input \( x \) of length \( n \) is at most
\[
2s(n)q(n)(1 - 2^{-q(n)}) + 2(2^{q(n)})q(n)(2^{q(n)}) = 2q(n)[s(n)(1 - 2^{-q(n)}) + 1],
\]
and this is polynomially bounded, provided that \( s(n) \) is polynomially bounded. By applying the Chernoff Bound on the tails of the binomial distribution to amplify the correctness probability of \( M_1 \) and \( M_2 \), we can indeed achieve error bound \( 2^{-q(n)} \) with polynomially many runs \( s(n) \). This shows that \( \text{RP} \cap \text{coRP} \) is contained in \( \text{ZPP} \).

Now let \( L \) be a language in \( \text{ZPP} \) and \( M \) be a probabilistic TM that accepts \( L \) and has expected (polynomial) running time \( q(n) \). Because \( \text{ZPP} \) is closed under complementation, it suffices to show that \( L \) is in \( \text{RP} \). Let \( M' \) be a probabilistic TM that, on input \( x \) of length \( n \), simulates \( M \) for time \( 4q(n) \). If, during this time, \( M \) accepts \( x \), then \( M' \) accepts \( x \) and halts; if, during this time, \( M \) rejects \( x \), or if \( M \) has neither accepted nor rejected during the first \( 4q(n) \) steps of the simulation, then \( M' \) rejects \( x \) and halts. The running time of \( M' \) is bounded by (the polynomial) \( 4q(n) \). If \( x \) is not in \( L \), then \( M' \) rejects \( x \). If \( x \) is in \( L \), then the probability that \( M' \) rejects \( x \) is exactly the probability that running \( M \) to completion on input \( x \) would take time more than \( 4q(n) \); that is the probability that a random variable attains a value that is more than 4 times its expected value, which, by Markov’s inequality, is at most \( 1/4 \).

(c) If \( \text{NP} = \text{P} \), then the entire \( \text{PH} \) is equal to \( \text{P} \). Because \( \text{BPP} \) is contained in the \( \text{PH} \) (by the Sipser-Gacs Theorem), it, too, is equal to \( \text{P} \).