Problem 1

Our instance checker I first runs the candidate algorithm M on the input matrices A and B to receive some matrix C as an answer. To check that indeed $AB = C$, it generates a random vector $r \in \{0,1\}^n$ and then verifies that $A(Br) = Cr$; if this is the case it accepts otherwise it outputs "fail". Since multiplication of a matrix by a vector requires $O(n^2)$ time, this verification can be done in $O(n^2 \log n)$ time (the $\log n$ term accounts for the time needed to read input bits via random access). For completeness, notice that if $AB = C$ the checker accepts with probability 1. If $AB \neq C$, then the matrix $AB - C$ has at least one nonzero entry, so for at least half the vectors $r \in \{0,1\}^n$, $(AB - C)r \neq 0$ and thus $\Pr[I^M(x) \in \{L(x), \text{“fail”}\}] \geq 1/2$. Repeating the procedure twice gives the desired error rate of 3/4.

Problem 2

(a) As pointed out in the hint, the prover in the interactive protocol for PSPACE languages can be realized by a polynomial space machine. (The verifier in this protocol asks for evaluations of an implicit polynomial; such evaluations can be computed in polynomial space.) Let L be a PSPACE-complete language. Here is a first attempt towards an instance checker for L. The instance checker I simulates the verifier; whenever the verifier wants to query the prover, the instance checker converts the query to an instance of L (this can be done efficiently since L is PSPACE-complete) and queries the oracle on this instance. If at the end of the interaction the verifier in the interactive protocol accepts, so does the instance checker; otherwise the instance checker outputs "fail".

By the completeness of the interactive protocol for PSPACE, it follows that if $x \in L$ and A solves L correctly on all inputs, then $I^A(x) = L(x)$ with probability one. However, when $x \notin L$, then $I^A(x) =$ "fail" with probability 3/4 for *every* algorithm A. This is not good because we want $I^A(x) = L(x)$ when A is a good algorithm for L.

To get around this problem, we use the fact that PSPACE is closed under complement, so I can run both the interactive protocols for L and for $\overline{L}$. More precisely, on input x, I does the following:

- Simulate the verifier in the interactive protocol for L, using the oracle as the prover. If the verifier accepts, accept.
- Simulate the verifier in the interactive protocol for $\overline{L}$, using the oracle as the prover. If the verifier accepts, reject.
- Otherwise, output "fail".

If the oracle A is a good algorithm for L, then $I^A(x)$ accepts when $x \in L$ (by completeness of the protocol for L) and rejects when $x \notin L$ (by completeness of tha protocol for $\overline{L}$). On the other hand, for every algorithm A, if $x \in L$ then the interactive protocol for $\overline{L}$ rejects with probability $\geq 3/4$, so $I^A(x) \notin \{L(x), \text{"fail"}\}$ with probability at most $1/4$. Similarly if $x \notin L$ then the interactive protocol for L rejects with probability $\geq 3/4$, so $I^A(x) \notin \{L(x), \text{"fail"}\}$ with probability at most $1/4$.

(b) Let $M_1, M_2 \ldots$ be an enumeration of polynomial time Turing machines, and let I be the instance checker for L. By definition, we have that for every A and x, $I^A(x) \notin \{L(x), \text{"fail"}\}$ with probability at most $1/4$. We can make this probability as small as 2^{-n^c} by running I $2^{O(n^c)}$ times and taking the plurality of the answers. (c is a sufficiently large constant we will specify later.)

Consider the following algorithm A: On input x, simulate in a dovetailing manner $I^{M_1}, I^{M_2}, \ldots$ (in stage i of the simulation, A runs i steps of $I^{M_1}(x), \ldots, I^{M_i}(x)$). If at any point some I^{M_i} returns an answer other than "fail", A outputs this answer and halts.

Since $L \in$ PSPACE, there exists an algorithm that decides L and runs in exponential time. Suppose this is the algorithm M_k. Then I^{M_k} also runs in exponential time, so after $2^{|x|^d}$ steps (where d is some constant that depends on k but not on x), $A(x)$ will have completed the simulation of $I^{M_k}(x)$, and by the completeness of I, will have output an answer with probability 1. Thus the running time of A on inputs of length n is at most 2^{n^d}.

We choose $c = d + 1$. Since in time 2^{n^d} the algorithm A can make at most 2^{n^d} calls to I, and for each call to I we have that $\Pr[I^{M_i}(x) \notin \{L(x), \text{"fail"}\}] \leq 2^{-n^c}$, by a union bound we have that with probability $\geq 3/4$, $I^{M_i}(x)$ never outputs $\overline{L}(x)$ in any of the calls made by A, so with probability $\geq 3/4$ A itself never outputs $\overline{L}(x)$.

Assuming this is the case, let M_i be an algorithm that decides L. Let x be a sufficiently long input for M_i. If $M_i(x)$ halts within t steps, then A will have finished simulating $I^{M_i}(x)$ within $p_i(|x|) \cdot t^3$ steps, where p_i is some polynomial that depends on i but not on x or t. By completeness of the instance checker, $I^{M_i}(x) = L(x)$, so $A(x)$ will also output $L(x)$.

Problem 3

(a) It is not hard to verify that the family of permanent polynomials $\{p_1, p_2, \ldots\}$ satisfies the given system of equations. The first equation indicates the permanent expansion by minors: We can write

$$\begin{aligned}
\mathrm{per}_n(x_{ij})_{1\leq i,j\leq n} &= \sum_{\sigma} \prod_{i=1}^{n} x_{i,\sigma i} \\
&= \sum_{k=1}^{n} \sum_{\sigma:\sigma(1)=k} x_{1k} \cdot \prod_{i=2}^{n} x_{i,\sigma i} \\
&= \sum_{k=1}^{n} \mathrm{per}_{n-1}(x_{ij})_{1\leq i,j\leq n, i\neq 1, j\neq k}.
\end{aligned}$$

The second equation indicates that if we replace the last column and row of an $n \times n$ matrix by zeros, except $y_{nn} = 1$, then we obtain the permanent of an $(n-1) \times (n-1)$ matrix.

$$\begin{aligned}
\mathrm{per}_n(y_{ij})_{1\le i,j\le n} &= \sum_{\sigma} \prod_{i=1}^{n} y_{i,\sigma i} \\
&= \sum_{\sigma:\sigma(n)=n} \prod_{i=1}^{n-1} y_{i,\sigma i} \\
&= \mathrm{per}_{n-1}(x_{ij})_{1\le i,j\le n-1}.
\end{aligned}$$

If $\{p_1, p_2, \dots\}$ and $\{p'_1, p'_2, \dots\}$ are two families of polynomials that satisfy the equations, it is immediate by induction on n that we must have $p_i = p'_i$ for all i.

(b) The instance checker I we are going to construct works in three stages. Suppose I is given as input an $n \times n$ matrix x and access to an oracle P.

- In the first stage I runs the local test A^P for degree n polynomials, viewing the candidate algorithm P as an oracle providing the value of a polynomial on the queried points. If P passes the local test then with high probability P computes correctly some degree n polynomial on 7/8 fraction of the points. Let p_n be such a polynomial. In particular, if P is the permanent polynomial per_n, then so is p_n.
- From p_n, we define the polynomials $p_{n-1}, p_{n-2}, \dots, p_1$ via the equation

$$p_{m-1}(x_{ij})_{1\le i,j\le m-1} = p_m(y_{ij})_{1\le i,j\le m}. \tag{1}$$

  We want to check that the polynomials $p_1, \dots, p_n$ satisfy the other equation that defines the permanent. To do this we invoke the randomized algorithm for polynomial identity testing on the input

$$p_m(x_{ij})_{1\le i,j\le m} - \sum_{k=1}^{m} x_{1k} \cdot p_{m-1}(x_{ij})_{1\le i,j\le m, i\ne 1, j\ne k}$$

  for every m between 1 and n. If any of the tests fail, I outputs "fail". If all of these identities hold, then by part (a) p_n must be the permanent polynomial per_n.

  Recall that the algorithm for polynomial identity testing works by evaluating the polynomial at a random input. To do this, we need to be able to evaluate p_m and p_{m-1} at inputs chosen by the identity testing algorithm. Evaluating p_m and p_{m-1} at some input in turn reduces to evaluating p_n at some other input (by equation (1)). Since p_n is 7/8-close to the oracle algorithm P, we can evaluate p_n at random inputs by using the reconstruction algorithm for P. (Recall that to evaluate $p_n(x)$, this algorithm chooses a random line through x and finds the value $p_n(x)$ by looking at values of P at other points on this line.)

  If $P = p_n = \mathrm{per}_n$, then all the identity tests will pass with probability 1. If $p_n \ne \mathrm{per}_n$, then at least one of the polynomials tested by the identity testing algorithm is nonzero, and (assuming the reconstruction algorithm works correctly, which happens say with probability $1 - O(1/n)$) the identity testing algorithm detects this with probability $1 - n/|\mathbb{F}| = 1 - O(1/n)$. Thus at the end of this stage, unless I outputs "fail", we know with high confidence that P is 7/8-close to the permanent polynomial.

- Finally, using the reconstruction algorithm for the permanent, I computes $p_n(x)$ using oracle access to P.