Question 1

Let $x_1, \dots, x_{10} \in \{0,1\}^{100}$ be 10 strings of 100 bits each. The distinctness function $DIST(x_1, \dots, x_{10})$ takes value 1 if all strings are distinct ($x_i \neq x_j$ when $i \neq j$) and 0 otherwise.

(a) Show that any deterministic read-once branching program for $DIST$ (that reads its input from left to right) must have width at least 2^{490}.

Solution: We claim that all *sets* S consisting of the first five inputs $\{x_1, x_2, x_3, x_4, x_5\}$ must lead to a different state in the middle layer. For contradiction, if two distinct sets $S = \{x_1, \dots, x_5\}$ and $S' = \{x'_1, \dots, x'_5\}$ lead to the same state then there must be some element x that is in say S but not S'. Consider a sequence $x_6, \dots, x_{10}$ that contains x but none of the elements in S'. Then $x_1, \dots, x_{10}$ aren't distinct but $x'_1, \dots, x'_5, x_6, \dots, x_{10}$ are, but the branching program produces the same answer to both inputs so it cannot compute $DIST$. Therefore the width must be at least the number of subsets of $\{0,1\}^{10}$ of size 5 or less which is at most 2^{492}. In fact 5 can be replaced with 9 leading to the stronger bound $\sum_{i=1}^{9} \binom{2^{100}}{i} > 2^{881}$.

(b) Let $h\colon \{0,1\}^{100} \to \{0,1\}^{10}$ be a random function. Show that with probability at least 95%, $DIST(h(x_1), \dots, h(x_{10})) = DIST(x_1, \dots, x_{10})$ for any fixed choice of inputs.

Solution: If $x_1, \dots, x_{10}$ are not distinct then $h(x_1), \dots, h(x_{10})$ are never distinct. If $x_1, \dots, x_{10}$ are distinct then $h(x_i) = h(x_j)$ with probability $1/2^{10}$ (as both values are random and independent) so by a union bound

$$\begin{aligned}\Pr[DIST(h(x_1), \dots, h(x_{10})) = 0] &= \Pr[\exists i < j : h(x_i) = h(x_j)] \\ &\leq \sum_{i<j} \Pr[h(x_i) = h(x_j)] = \binom{10}{2} \cdot 2^{-10} < 5\%.\end{aligned}$$

(c) Show that $DIST$ can be computed by a randomized read-once branching program of width at most 2^{200} with error at most 5%.

Solution: The branching program in question chooses a random h. After reading x_i it stores the value $h(x_i)$. In the end it compares the stored values $h(x_1), \dots, h(x_{10})$ and accepts iff they are all distinct. By part (b) the answer is correct on every input except with probablity at most 5%. The program needs 10 bits to store each of the values $h(x_1), \dots, h(x_{10})$ plus 100 bits to store x_i as it is being read so it can be implemented with 200 (in fact 190) bits of memory, or width 2^{200}.

In fact the analysis in part (b) works even if, instead of being truly random, h is the function $h(x) = (IP(x; r_1), \dots, IP(x, r_{10}))$ where $r_1, \dots, r_{10} \in \{0,1\}^{100}$ are random strings and IP is the inner product function from Lecture 3. This function is not random, but the only property of h used in part (b) was that $\Pr[h(x_i) = h(x_j)] = 2^{-10}$ when x_i and x_j are distinct and this still holds because $\Pr[IP(x_i, r) = IP(x_j, r)] = \Pr[IP(x_i + x_j, r) = 0] = 1/2$. This implementation only needs to store 10 bits of the input it is currently reading so the resulting branching program has width only 100.

Question 2

Let X be an n by n matrix and $f\colon \{0,1\}^{n^2} \to \{0,1\}$ be the function

$$f(X) = \begin{cases} 1, & \text{if } f \text{ has } \textit{exactly one} \text{ column consisting of zeros only,} \\ 0, & \text{otherwise.} \end{cases}$$

Determine the following quantities up to a constant factor (i.e., in $\Theta(\cdot)$ notation). Provide both upper and lower bound proofs.

(a) the deterministic query complexity $D(f)$

Solution: This is n^2 by an "adversary argument". Answer the queries of the decision tree by zeros, until a whole column is queried, in which case the last column query is answered by 1. If the decision tree has depth strictly less than n^2 the queried part of the input X is consistent both with the possibilities $f(X) = 0$ and $f(X) = 1$, so the decision tree cannot compute f on all inputs.

(b) the exact degree $\deg(f)$ when f is viewed as a real-valued polynomial

Solution: This is also n^2, giving also an alternative proof of part (a). We will represent the polynomial as a function from $\{0,1\}^n \to \{0,1\}$ for convenience as this does not affect the degree. Then $f(X) = g(h(X^1), \ldots, h(X^n))$, where $X^1, \ldots, X^n$ are the columns of X, h is the "zeros only" function, and g is the "exactly one one" function. The unique polynomial representations of h and g are

$$h(x_1, \ldots, x_n) = (1 - x_1) \cdots (1 - x_n) \qquad g(y_1, \ldots, y_n) = \sum_{i=1}^{n} y_i \prod_{j \neq i} (1 - y_j).$$

Both h and g contain the degree-n monomials $x_1 \ldots x_n$ and $(-1)^{n-1} n y_1 \cdots y_n$, respectively, so their composition f must contain the degree-n^2 monomial $\prod_{i,j=1}^{n} X_{ij}$.

(c) the sensitivity $\text{sens}(f)$

Solution: This is $2n$. If X has exactly two all-zero columns, then changing any of the $2n$ entries in these columns flips the value of f showing that the sensitivity is at least $2n$. We argue it is at most $2n$ by cases. Matrices with 3 or more all-zero columns are insensitive. If there are exactly two, only the $2n$ entries in those two can change the value of f from 0 to 1. If there is exactly one all-zero column, then the entry can be changed from 1 to zero either by destroying this column or creating a new all-zero column. There are n choices for the first possibility and at most $n - 1$ for the second as the only way to create an all-zero column is to flip a 1-entry in it provided it is unique, for a total of at most $2n - 1$. Finally, if there are no all-zero columns, there can be at most n variables that can be flipped to create one.

(d) **(Extra credit)** the Monte Carlo randomized query complexity $R_{1/3}(f)$

Solution: It should be $\Omega(n^2)$. Let X_1 be a random matrix whose columns are random independent vectors with exactly one 1, and X_0 be a random matrix like X_1 except a random 1-entry is converted to a 0. I think it can be shown that the advantage of a decision tree that makes say $n/100$ queries is at most $1/3$ but the analysis I have in mind is a bit complicated.

(e) **(Mini-research project)** the quantum query complexity $Q_{1/3}(f)$

Question 3

A *solution generator* for a search relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ is an algorithm G that on input (x, ℓ) outputs the ℓ-th lexicographically smallest y such that $(x, y) \in R$, and the special symbol $\perp$ if such a y does not exist. For example, if $(0,0),(0,10),(0,111) \in R$ but $(0,y) \notin R$ for all other y then $G(0,1) = 0, G(0,2) = 10, G(0,3) = 111$, and $G(0,\ell) = \perp$ for all other ℓ. We say that a solution generator is *efficient* if its running time is polynomial in $|x|$ and ℓ.

(a) Prove that the search relation $R_{\text{DNF}} = \{(\phi, y) : \phi \text{ is a DNF such that } \phi(y) = 1\}$ (a DNF is an OR of ANDs of literals) has an efficient solution generator.

Solution: For each term t of the DNF let S_t be the set of the ℓ lexicographically smallest assignment that satisfy t. Once the variables in t are fixed to their satisfying value the rest of the assignment can be arbitrary so S_t can be listed in time $O(n\ell)$. The ℓ-th smallest assignment to ϕ must then belong to one of the sets $S_1, \dots, S_m$, where m is the number of terms. The efficient solution generator lists the sets S_t (in time $O(n\ell)$), sorts them (in time at most $O(mn\ell \log m\ell)$)) and then outputs the ℓ-th assignment.

An alternative solution is the following recursive algorithm $LIST(\phi, \ell)$ that lists the first ℓ solutions (or fewer if there aren't ℓ): If ϕ is unsatisfiable or $\ell = 0$ output the empty list. Otherwise, run $LIST(\phi_0, \ell)$ followed by $LIST(\phi_1, \ell - \ell_0)$, where ϕ_b is ϕ restricted to $x_1 = b$ and ℓ_0 is the number of solutions produced by $LIST(\phi_0, \ell)$. The running time of $LIST$ satisfies the recurrence $T(n, \ell) \le T(n-1, \ell_0) + T(n-1, \ell-\ell_0) + t$ where t is the time to check if ϕ is satisfiable (which is polynomial in $|\phi|$). By induction on n it follows that $T(n, \ell) \le n\ell t$.

(b) Prove that if the search relation $R_{\text{CNF}} = \{(\phi, y) : \phi \text{ is a CNF such that } \phi(y) = 1\}$ (a CNF is an AND of ORs of literals) has an efficient solution generator then P = NP.

Solution: Under this assumption SAT is in P because on input $(\phi, 1)$ if the solution generator for R_{CNF} output a satisfying assignment then ϕ is satisfiable and if it outputs $\perp$ it isn't. Since SAT is NP-complete, P must equal NP.

(c) Prove that if P = NP then every NP search relation has an efficient solution generator.

Solution: In Lecture 7 we described an algorithm that outputs *a* solution for CSAT given that one exists. In fact the algorithm we described outputs the lexicographically smallest solution. Moreover, the reduction from problems in NP to CSAT preserves the solution space, so the lexicographically smallest solution to any NP problem can be found in polynomial time.

Given an NP-search relation R, consider the relation

$$R_\ell = \{((x, 1^\ell), (y_1 \cdots y_\ell)) \colon R(x, y_1) \text{ and } \cdots \text{ and } R(x, y_\ell) \text{ and } y_1 < \cdots < y_\ell\}$$

where $<$ is the lexicographic ordering. This is also an NP-relation since the condition can be verified by running ℓ copies of the verifier for R and checking the ordering, which takes time $O(\ell + |x|)$. We added the portion 1^ℓ to the input so the running time stays polynomial in the length of the input $(x, 1^\ell)$ to R_ℓ. We can therefore find the lexicographically smallest solution to R_ℓ, which equals the ℓ lexicographically smallest solutions of R, in time polynomial in $|x| + \ell$.

An alternative solution is to run the same algorithm as in part (a) applied to the CSAT instance C. Checking that C is satisfiable can be done using the assumed polynomial-time algorithm for CSAT so the algorithm is efficient assuming P = NP.