1. **[PAC-learning of small OR-functions]** Suppose the target function is a disjunction (OR-function) of \( r \) out of \( n \) boolean variables, where \( r \) is much less than \( n \). For example, perhaps only \( \sqrt{n} \) of the \( n \) variables are used in the target \( (r = \sqrt{n}) \). The list-and-cross-off algorithm for learning an OR function that we gave in class might produce a hypothesis of size \( O(n) \), so its sample size for PAC-learning would be \( \tilde{O}(n/\epsilon) \). Alternatively, we could try all \( O(n^r) \) possible OR-functions of size \( \leq r \) and pick one consistent with our training sample. Since \( \log |H| = r \log n \), this would require only \( \tilde{O}((r \log n)/\epsilon) \) samples but takes time exponential in \( r \).

**Your job:** Give a polynomial-time algorithm that guarantees to find an OR of at most \( O(r \log m) \) variables that is consistent with the training data, where \( m \) is the number of training examples. So this is not quite finding the *smallest* OR-function, but it’s close. Describe a variant of your algorithm (also running in polynomial time) that finds an OR of only \( O(r \log (1/\epsilon)) \) variables, with error at most \( \epsilon/2 \) on the training data. Note that by Occam + Chernoff bounds, the latter algorithm requires sample size at most \( O(\frac{1}{\epsilon}((r \log 1/\epsilon) \log n + \log 1/\delta)) \) to learn in the PAC model, so its sample complexity is just very slightly worse than the exponential-time algorithm that tries all \( O(n^r) \) OR-functions of size \( \leq r \).

Hint: think about the greedy set-cover algorithm (look it up if you haven’t seen it).

2. **[Uniform distribution learning of DNF Formulas]** Give an algorithm to learn the class of DNF formulas having at most \( s \) terms over the uniform distribution on \( \{0, 1\}^n \), which has sample size polynomial in \( n \) and \( s \), and running time \( n^{O(\log(s/\epsilon))} \). So, your algorithm matches the SQ-dimension lower bounds.

Hint: Think of your algorithm for Problem 1.

Note: your solution requires that data come from the uniform distribution. The best algorithm known for learning polynomial-size DNF formulas over general distributions has running time roughly \( 2^{O(n^{1/3})} \) [Klivans-Servedio].

3. **[SQ learning Decision Lists and Trees]** Give an algorithm to learn the class of decision lists in the SQ model (and argue correctness for your algorithm). Your algorithm should work for any distribution \( D \) (not just the uniform distribution). Be clear about what specifically the queries \( \chi \) are and the tolerances \( \tau \). Remember, you are not allowed to ask for conditional probabilities like \( \Pr[A|B] \) but you can ask for \( \Pr[A \land B] \).
So, combined with your results from Homework 1, this gives an algorithm for learning decision trees of size $s$ in the SQ model with $n^{O(\log s)}$ queries of tolerance $1/n^{O(\log s)}$, matching our SQ-dimension lower bounds.

Note: this problem can be tricky. In particular, it is possible to create a distribution $D$ over $\{0, 1\}^n$ and a target decision list $c$ with the following properties:

(a) $\Pr_D[c(x) = 1] = 1/2$.
(b) For all $1 \leq i \leq n$, either $\Pr_D[x_i = 1] \leq 2^{-n/2}$ or else $\Pr_D[c(x) = 1|x_i = 1] = 1/2$.

In particular, no variable is noticeably correlated with the target! (Any variable either is almost never 1 or else is completely uncorrelated with the target). So, an algorithm that tries to find $x_i, y, b$ such that both (a) $\Pr[c(x) = y|x_i = b]$ is large and (b) “$x_i = b$” happens with noticeable probability is going to have trouble. In fact, this difficulty is one reason that there is no known analog of Problem 1 for decision lists. Instead, think about building on the algorithm from the very first lecture.