• You must type your solutions. \LaTeX, Microsoft Word, and plain ascii are all acceptable. Submit your solutions via email to cs221-hw@seas.harvard.edu. If you use \LaTeX, please submit both the compiled file (.pdf) and the source (.tex). Please name your files PS4-yourlastname.*.

• Strive for clarity and conciseness in your solutions, emphasizing the main ideas over low-level details. Do not despair if you cannot solve all the problems! Difficult problems are included to stimulate your thinking and for your enjoyment, not to overwork you. *'ed problems are extra credit.

Problem 1. (one-sided error vs. two-sided error) Show that if $\mathsf{NP} \subseteq \mathsf{BPP}$, then $\mathsf{NP} = \mathsf{RP}$.

Problem 2. (Cook reductions to promise problems) Note that for a promise problem $\Pi$, “running an algorithm with oracle $\Pi$” is not in general well-defined, because it is not specified what the oracle should return if the input violates the promise.\footnote{A similar issue comes up with problems where there are multiple valid answers on a given input, such as search or approximation problems. Again, in such cases, we should require that the algorithm works correctly for every oracle that solves the problem.} Thus, when we say that a problem $\Gamma$ can be solved in polynomial time with oracle access to $\Pi$, we mean that there is a polynomial-time oracle algorithm $A$ such that for every oracle $O: \{0,1\}^* \rightarrow \{0,1\}$ that solves $\Pi$ (i.e. $O$ is correct on $\Pi_Y \cup \Pi_N$), it holds that $A_O$ solves $\Gamma$.

1. Let $\Pi$ be the promise problem

\[\Pi_Y \overset{\text{def}}{=} \{ (\varphi, \psi) : \varphi \in \mathsf{SAT}, \psi \notin \mathsf{SAT} \}\]

\[\Pi_N \overset{\text{def}}{=} \{ (\varphi, \psi) : \varphi \notin \mathsf{SAT}, \psi \in \mathsf{SAT} \}\]

Show that $\Pi \in \mathsf{prNP} \cap \mathsf{prcoNP}$ but $\mathsf{SAT} \in \mathsf{prP}^\Pi$.

This implies that $\mathsf{prNP} \subseteq \mathsf{prP}^{\mathsf{prNP} \cap \mathsf{prcoNP}}$. Note that an analogous inclusion seems unlikely for language classes, since $\mathsf{P}^{\mathsf{NP} \cap \mathsf{coNP}} = \mathsf{NP} \cap \mathsf{coNP}$, as shown in an earlier section.

2. (*) Show that $\mathsf{prBPP} \subseteq \mathsf{prRP}^{\mathsf{prRP}}$, and thus $\mathsf{prRP} = \mathsf{prP}$ iff $\mathsf{prBPP} = \mathsf{prP}$. (Hint: look at the proof that $\mathsf{BPP} \subseteq \mathsf{PH}$.)
Problem 3. (**#Matchings** and **#Independent Sets**)  

1. A matching in a graph is a set $S$ of edges such that every vertex touches **at most one** edge in $S$ (as opposed to exactly one, as required in a perfect matching). Show that **#Matchings**, the problem of counting all the matchings in a graph, is **#P**-complete. (Hint: reduce from **#Perfect Matchings**. Given a graph $G$, consider the graph $G_k$ obtained by attaching $k$ new edges to each vertex of $G$. $G_k$ has $n + nk$ vertices, where $n$ is the number of vertices in $G$. Show that the number of perfect matchings in $G$ can be recovered from the number of matchings in each of $G_0, \ldots, G_n$.)

2. An independent set in a graph $G$ is a set $S$ of vertices such that no two elements of $S$ are connected by an edge in $G$. Prove that **#Independent Sets**, the problem of counting the number of independent sets in a graph, is **#P**-complete.

3. Prove that a fully polynomial randomized approximation scheme for **#Matchings** implies a fully polynomial almost-uniform sampler for **Matchings**. (This is the converse of what we showed in class.)

4. Show that approximating **#Independent Sets** to within any constant factor is **NP**-hard. In contrast, there are a fully polynomial randomized approximation schemes known for **#Perfect Matchings** and **#Matchings**.

Problem 4. (**parallel search vs. decision**)  

1. Recall that we can solve the SAT-SEARCH problem in polynomial time given an oracle for deciding SAT. Note that this reduction algorithm makes **adaptive** queries to the SAT oracle, i.e. its $i$'th query depends on the answers to its first $i - 1$ queries. Show that the reduction can be made nonadaptive if we allow it to be randomized. (Hint: Use Valiant-Vazirani)

2. Suppose that there is a $1/poly(n)$-hard **NP** search problem $S$ (see PS0 for definition). That is, there is a polynomial $p$ such that for every probabilistic polynomial-time machine $A$ and sufficiently large $n$,  
\[
\Pr_{x \leftarrow \{0,1\}^n} [A(x) \not\in S(x)] \geq 1/p(n).
\]
Show that there is a $1/poly(n)$-hard language in **NP**.