Problem Set 4

- Due Date: **26th May 2021**
- Turn in your problem sets electronically (pdf or text file) on Acadly.
- Collaboration is encouraged, but all writeups must be done individually and must include names of all collaborators.
- Referring to sources other than the text book and class notes is strongly discouraged. But if you do use an external source (eg., other text books, lecture notes, or any material available online), ACKNOWLEDGE all your sources (including collaborators) in your writeup. This will not affect your grades. However, not acknowledging will be treated as a serious case of academic dishonesty.
- The points for each problem are indicated on the side. There are **6 questions** with a total of **100 points** in this problem set.
- Some of the questions are broken-down in to multiple subdivisions to guide you towards a solution. Feel free to use the earlier subdivisions for later ones, even if you haven't solved it (yet).
- Be clear in your writing.

Question 1 (Parallel decision vs search)**.** (10)

Recall that we can obtain a satisfying assignment for a Boolean formula (if one exists) in polynomial time given an oracle for deciding SAT (using downward self-reducibility of SAT). However, 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 reduction.]

Question 2 (#P-completeness)**.** $(3 + 3 + (3 + 7 + 6) + 3)$

Recall the definition of #P-completeness. A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ in #P is said to be #P-complete if for every $g \in \#\mathsf{P}$, there exists a deterministic polynomial time oracle Turing machine M such that for all $x \in \{0,1\}^n$, $M^f(x) = g(x)$.

(a) Prove that for any constant $c > 0$, obtaining a c-approximation for # SAT (where the input is an arbitrary formula) is NP-hard. (Recall that we say a TM M computes a c-approximation for this function if, for any input φ, we have

$$\frac{1}{c} \cdot \#\,\mathrm{SAT}(\varphi) \leq M(\varphi) \leq c \cdot \#\,\mathrm{SAT}(\varphi).$$

We almost did this in class but do spell it out in this answer.)

(b) Let # DNF be the function that, when provided an suitably encoded DNF as input returns the number of satisfying assignments of the DNF. Show that # DNF is #P-complete

(c) In this part, we will see that we can get an $(1+\varepsilon)$-approximation for # DNF in polynomial time (without the need for any NP-oracle). Let φ be the input DNF on n variables.

(i) Consider the following naïve algorithm:

> Randomly sample t strings $x^{(1)}, \ldots, x^{(t)} \in \{0,1\}^n$. Compute the fraction p of these strings that satisfy φ. Output $p \cdot 2^n$.

How large should t be in order to obtain a $(1+\varepsilon)$-approximation for #DNF?

(ii) Let $\varphi = T_1 \vee T_2 \vee \cdots \vee T_m$ where each T_i is a term (AND of literals). Show that size of the set

$$S' = \{(x, i) \in \{0,1\}^n \times [m] \ : \ x \text{ satisfies term } T_i\}$$

can be computed efficiently. Furthermore, give a polynomial time randomized algorithm to sample uniformly at random from the above set S'.

(iii) Consider the following algorithm:

> Pick t independent uniform samples from S' — let the *multiset* of these samples be $\{(x^{(1)}, i_1), \ldots, (x^{(t)}, i_t)\}$.
> Compute the fraction p of these samples $(x^{(i)}, t_i)$ that satisfy the property that i is the index of the *first* term that $x^{(i)}$ satisfies.
> Return $p \cdot |S'|$.

How large should t be so that the above algorithm yields a $(1+\varepsilon)$-approximation for #DNF?

(d) Part (a) claims that any c-approximation for #SAT is NP-hard. On the other hand, part (c) claims that #DNF has a polynomial time $(1+\varepsilon)$-approximation algorithm. Both of these problems are #P-hard. Why does this not show that NP $\subseteq$ BPP?

Question 3. (7 + 13)

An arithmetic formula is defined as a tree with internal gates labeled by $+$ and $\times$ that have the obvious semantics of adding and multiplying the values computed by its children respectively. The leaves of the tree are either labeled by input variables x_i or constants -1 or 0 or 1. Note that the *root* of the tree naturally computes a polynomial $f(x_1, \ldots, x_n)$ with integer coefficients.

(a) Consider the following language:

$$L = \{(C_1, C_2) \ : \ C_1 \text{ and } C_2 \text{ compute the same polynomial}\}.$$

Show that $L \in$ coRP (the algorithm must run in polynomial time in the description of C_1 and C_2, which is roughly the number of nodes in the two circuits).

(b) If the permanent had a polynomial time algorithm, then we know that $\mathsf{P}^{\#\mathsf{P}} = \mathsf{P}$. Suppose instead that we are only todl that there is is a family of polynomial-size arithmetic formulas $\{C_n\}_{n=1}^{\infty}$ such that C_n computes the permanent of $n \times n$ matrices. Further assume that RP = P. Show that these two hypotheses imply that $\mathsf{P}^{\#\mathsf{P}} = \mathsf{NP}$.

[Hint: How do you verify if a certain arithmetic formula family is actually computing the permanent? The downward self-reducibility of the permanent should help you here.]

Question 4 (Downward self-reducibility (Problem 8.9 in Arora-Barak))**.** (10)

Formally, a language L is said to be *downward self-reducible* if there is a polynomial time deterministic oracle-TM M that, for any n and $x \in \{0,1\}^n$, we have that $M^{L_{n-1}}(x) = L(x)$ where $L_{n-1} = \{y \in L \ : \ |y| \leq n-1\}$. That is, the machine M can correctly decide membership of strings of length n when given oracle access to strings of smaller length in the language.

Show that *any* downward self-reducible language L is in PSPACE.

Question 5 (#L problems). (10)

The class #P could have been defined in two equivalent ways.

Option 1: A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #P if there is a non-deterministic Turing machine M_f that on input x of length n uses poly(n) time and is such that the number of accepting paths of $M_f(x)$ equals $f(x)$.

Option 2: A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #P is there is a relation $R(\cdot,\cdot)$ that is computable in polynomial time and a polynomial p such that $f(x)$ equals $|\{y : R(x,y) \text{ is true, and } |y| \leq p(|x|)\}|$.

(You may check for yourself that these two definitions are indeed equivalent.)

In this problem, we will show that the corresponding two definitions are very different if one considers counting problems in logspace. Consider the following two definitions of log-space counting problems.

#L$_1$: A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #L$_1$ if there is a non-deterministic Turing machine M_f that on input x of length n uses $O(\log n)$ space and is such that the number of accepting paths of $M_f(x)$ equals $f(x)$.

#L$_2$: A function $f : \{0,1\}^* \to \mathbb{Z}^{\geq 0}$ is in #L$_2$ is there is a relation $R(\cdot,\cdot)$ that is computable in logarithmic space and a polynomial p such that $f(x)$ equals $|\{y : R(x,y) \text{ and } |y| \leq p(|x|)\}|$.

Prove that all functions in #L$_1$ can be computed in polynomial time whereas #L$_2$ equals #P.

[Hint: It may be useful to recall 3SAT has a verifier $M(\cdot,\cdot)$ that is a logspace machine.]

Question 6 (Promise problems). (4 + 3 + 9 + 6 + 3)

(a) Show that $\mathsf{P}^{\mathsf{NP}\cap\mathsf{coNP}} = \mathsf{NP} \cap \mathsf{coNP}$.

Recall the definition of a promise problem. Note that for a promise problem Π, "running an algorithm with oracle Π" is not in general well-defined, because it is not specified what the oracle should return if the input violates the promise. Thus, when we say that a problem Γ can be solved in polynomial time with oracle access to Π, we mean that there is a polynomial-time oracle algorithm A such that for *every* oracle $O : \{0,1\}^* \to \{0,1\}$ that solves Π (i.e. O is correct on $\Pi_Y \cup \Pi_N$), it holds that A^O solves Γ. Let Π be the promise problem

$$\Pi_Y \stackrel{\text{def}}{=} \{(\varphi,\psi) : \varphi \in \text{SAT}, \psi \notin \text{SAT}\}$$
$$\Pi_N \stackrel{\text{def}}{=} \{(\varphi,\psi) : \varphi \notin \text{SAT}, \psi \in \text{SAT}\}$$

(b) Show that $\Pi \in \mathsf{prNP} \cap \mathsf{prcoNP}$

(c) Show that $\text{SAT} \in \mathsf{P}^{\Pi}$.

[Hint: You might have to query the oracle Π more than once, even a superconstant number of times. Downward self-reducibility might help you again.]

Remark. In fact, the above fact can be extended to say that for any promise problem Γ in prNP, we have that $\Gamma \in \mathsf{prP}^{\Pi}$.

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 the earlier part.

(d) Show that $\mathsf{prBPP} \subseteq \mathsf{prRP}^{\mathsf{prRP}}$.

[Hint: You might find the proof of $\mathsf{BPP} \subseteq \Sigma_2^P$ useful.]

(e) Use the above to show that $\mathsf{prRP} = \mathsf{prP}$ if and only if $\mathsf{prBPP} = \mathsf{prP}$.
