Problem Set 2

- Due date: **9 Mar, 2024** (released on 22 Feb, 2024; last addition on 22 Feb, 2024).
- The points for each problem is indicated on the side.
- More problems will be released gradually as we cover more material in class. At the moment, the total for this set is **45** points.
- The problem set has a fair number of questions so please do not wait until close to the deadline to start on them. Try and do one question every couple of days.
- Turn in your problem sets electronically (PDF; either LaTeXed or scanned etc.) on Piazza via private post.
- Collaboration with other students taking this course is encouraged, but collaboration with others is not allowed. Irrespective of this, all writeups must be done individually and must include names of all collaborators (if any).
- 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.
- Be clear in your writing.

1. **[Baker-Gill-Solovay for NP and coNP]** (10)

   Show that there is a language A such that $\mathsf{NP}^A \neq \mathsf{coNP}^A$.

   [Hint: It might be useful to remember that coNP rejects if *any* of its computational paths reject. Modify the construction seen in class appropriately.]

2. **[Cook-Levin in the presence of oracles]** (7 + 8)

   A natural question is whether the Cook-Levin reduction continues to hold even in the presence of oracles. Turns out, the answer is 'Yes, and no' — it depends on how precisely the question is posed.

   (a) Let A be an arbitrary language. Define a suitable *relatived* version of CircuitSAT, called say CircuitSAT^A, and show that it is NP^A-complete under polynomial time many-one reductions.

   (Here the reductions are standard polynomial time many-one reductions and does not use access to the oracle A.)

   (b) Show that there is a language A and another language L_A such that $L_A \in \mathsf{NP}^A$ but there is no polynomial-time oracle TM M such that M^A is a reduction from L_A to CircuitSAT.

   (That is, we cannot hope to reduce L_A to CircuitSAT where the oracle queries are only made in the 'reduction'.)

   [Hint: Recall the oracle we used in the Baker-Gill-Solovay theorem. There, we had to diagonalise against polynomial time TMs computing a decision problem. Can you modify it suitably to diagonalise against polynomial time oracle-reductions?]

3. **[Classification of problems]** (10)

   For each of these problems, mention if they are in P, NP, coNP, $\mathsf{P}^{\mathsf{SAT}}$, $\mathsf{NP}^{\mathsf{SAT}}$, $\mathsf{coNP}^{\mathsf{SAT}}$.

   (a) L_1 is consists of the set of representations of formula Φ such that Φ is satisfiable and the lexicographically largest satisfying assignment for Φ has the last variable set to 1. (We are implicitly assuming that there is some ordering on the variables, say $x_1, \ldots, x_n$ and thus assignments are n-length binary strings. If you want to think of these as n-bit integers, we are asking if 'largest satisfying integer' is odd.)

   (b) L_2 consists of a graph G and a subset of vertices S defined as follows. $(G, S) \in L_2$ is every 2-colouring of the vertices in S extends to a valid 3-colouring in G.

   (c) L_3 consists of all graphs G, with integer edge weights, such that the shortest travelling salesman route (a path that starts at some vertex v and visists every other vertex at least once) has length divisible by 42. (It might help to know that finding the length of the shortest travelling salesman route is NP-complete.)

   For each of the above problems, say whether you think the problem is complete for that class (under polynomial time many-one reductions).

4. **[Mahoney's theorem]** (10)

   Prove that if there is a sparse language L that is NP-hard, then $\mathsf{P} = \mathsf{NP}$.

   [Hint: You will have to follow the proof that we had in class, but you need a way to 'amplify' the number of satisfiable formulas in your bag. That is, can you do some operation on your bag of formulas so that, if you had even one satisfiable formula in your bag, you now have *many* of them after this operation?]