CS 278: Computational Complexity Theory
Homework 1

Due: September 26 2025

Fall 2025

Instructions:

- Collaboration is allowed but solutions must be written independently.
- Please write your solutions in a LaTeX document.
- Please submit your solutions via an email to lijiechen@berkeley.edu, the subject line should be "CS 278: Homework 1 – [Your Name]".
- Please Use "CS 278: Homework 1 – [Your Name].pdf" as the name of your homework.
- Please submit your solutions by 11:59PM on September 26, 2025, Pacific Time.
- Late submissions get a penalty of **10%** per day, consult the lecturer if you need extensions. (i.e., being late by 3 days gets $0.9^3 = 0.729$ fraction of the score.).
- The maximum score of this homework is 160. There are 4 problems, and each problem is worth 40 points. If you get n points, your score for this homework is

$$a_1 = \frac{n}{100} \times 12.5$$

- Let a_1, a_2, a_3, a_4 be the scores for the 4 homeworks, your final grade of homework is $\min(a_1 + a_2 + a_3 + a_4, 50)$.
- In other words, you don't have to solve all the problems to get a perfect score on homeworks.

1 Problem 1: Non-deterministic time hierarchy theorem with bounded guess, revisited

In the class, we proved the following theorem:

Theorem 1 (Non-deterministic time hierarchy theorem with bounded guess)**.** *Let $T, G, W : \mathbb{N} \to \mathbb{N}$ be time-constructible functions such that $G(n) = o(T(n))$ and $W(n) = o(n)$. Then there is a language $L \in NTIME[T(n)]$ but L is almost-everywhere separated from $NTIMEGUESS[G(n), W(n)]$.*

Part (a). 20 pts Explain why the almost-everywhere separation against NTIMEGUESS$[T(n), n/10]$ does not work for NTIME$[T(n)]$, which part of the proof fails?

Part (b). 20 pts Strengthen the proof to show an almost-everywhere separation between NTIMEGUESS$[T(n), n]$ and NTIMEGUESS$[G(n), W(n)]$?

2 Problem 2: Robustly-often NTIME Hierarchy (40 pts)

We mentioned that the non-deterministic time hierarchy theorem only works either infinitely often, or almost everywhere, but only against the weak class NTIMEGUESS$[T(n), n/10]$.

Problem 1 asks you to prove the following theorem, which shows that it is possible to have a separation that is stronger than infinitely often (but weaker than almost everywhere), that holds for the general class NTIME$[T(n)]$.

Theorem 2. *Let $T(n) = n^K$ be a polynomial where $K \in \mathbb{N}$ is a constant. There is a language $L \in$ NTIME$[T(n)^2]$ such that for every $L' \in$ NTIME$[T(n)]$, for every sufficiently large $n_0 \in \mathbb{N}$, there exists an $n \in [n_0, T(n_0)^{1.5}]$ such that $L_n \neq L'_n$, here L_n denotes the restriction of L to input length n ($L_n = \{x \in L \mid |x| = n\}$).*

Hint 1. *The issue of applying the proof for NTIMEGUESS$[T(n), n/10]$ to NTIME$[T(n)]$ is that the hard machine is going to take the witness as part of the n-bit input, NTIME$[T(n)]$ has $T(n)$ bit witnesses, so it's impossible to include those in the n-bit input.*

But you may be able to deal with that by using the ideas from the original proof of NTIME hierarchy theorem!

3 Problem 3: Refuter for Theorem 1

Theorem 1 implies that, for the corresponding hard language $L \in \mathrm{NTIME}[T(n)]$, for every $\mathrm{NTIMEGUESS}[G(n), W(n)]$ machine M, there exists an integer $N_M \in \mathbb{N}$ such that for all $n \geq N_M$, there exists an input $x_n \in \{0,1\}^n$ such that $L_n(x_n) \neq M(x_n)$.

For Problem 3, to make things easier, we will assume that both $T(n)$ and $G(n)$ from Theorem 1 are polynomials in n.

Your task is to construct a "refuter" for Theorem 1, that is, a machine R that, it takes the description of a $\mathrm{NTIMEGUESS}[G(n), W(n)]$ machine M, as well as an input length n as input, and outputs a string $x_n = R(\langle M \rangle, n) \in \{0,1\}^n$ such that $L(x_n) \neq M(x_n)$, for sufficiently large $n \geq N_M$.

In a sense, we are asking to make the proof of Theorem 1 "constructive", in the sense that not only we want these $x_n \in \{0,1\}^n$ to exist, but we also want to be able to construct them by an explicit algorithm.

Your algorithm R should be an NP-oracle polynomial time deterministic machine. That is, it can make NP-oracle queries to some oracle $\mathcal{O} \in \mathrm{NP}$, and it can make polynomial number of queries to $\mathcal{O}$, and runs in deterministic polynomial time.

Part (a). 20 pts Suppose you are given query access to a list $a_1, a_2, \ldots, a_N$ of N integers, and you are promised that $a_1 \neq a_N$. Design a deterministic algorithm that finds an index i such that $a_i \neq a_{i+1}$, using at most $O(\log N)$ queries to the list.

Part (b). 20 pts Construct the required refuter algorithm R.

4 Relativization Barrier for P vs BPP

We now explore the relativization barrier for the P vs BPP problem. First, let's recall the definitions of these complexity classes.

We say a langauge L is in P if there exists a deterministic polynomial-time Turing machine M such that for all x:

- If $x \in L$, then $M(x) = 1$
- If $x \notin L$, then $M(x) = 0$

We say a langauge L is in BPP if there exists a *deterministic* polynomial-time Turing machine M and a polynomial $p\colon \mathbb{N} \to \mathbb{N}$ such that for all x:[1]

- If $x \in L$, then $\Pr_{r \in \{0,1\}^{p(|x|)}}[M(x,r) = 1] \geq 2/3$
- If $x \notin L$, then $\Pr_{r \in \{0,1\}^{p(|x|)}}[M(x,r) = 1] \leq 1/3$

Let $\mathcal{O}\colon \{0,1\}^* \to \{0,1\}$ be an oracle. We can define the classes $\mathrm{P}^{\mathcal{O}}$ and $\mathrm{BPP}^{\mathcal{O}}$ analogously, by changing the machine M from definition to $\mathcal{O}$-oracle Turing machine $M^{\mathcal{O}}$.

Part (a). 15 pts Show that there exists an oracle $\mathcal{O}_1$ such that $\mathrm{P}^{\mathcal{O}_1} = \mathrm{BPP}^{\mathcal{O}_1}$.

Part (b). 15 pts Show that there exists an oracle $\mathcal{O}_2$ such that $\mathrm{P}^{\mathcal{O}_2} \neq \mathrm{BPP}^{\mathcal{O}_2}$.

Part (c). 10 pts Show that there exists an oracle $\mathcal{O}_3$ such that $\mathrm{P}^{\mathcal{O}_3} = \mathrm{BPP}^{\mathcal{O}_3}$, yet $\mathrm{P}^{\mathcal{O}_3} \neq \mathrm{NP}^{\mathcal{O}_3}$.

[1] Note that here M itself is deterministic, the randomness is over the second input $r \in \{0,1\}^{p(|x|)}$.