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$[G(n), W(n)]$ proved above does not work for NTIME$[G(n)]$, which part of the proof fails? (i.e., if you attempt to use the same proof to prove that NTIME$[T(n)]$ is almost-everywhere separated from NTIME$[G(n)]$, 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 Solution to Problem 1

2.1 Part (a)

Note that in the proof, a cruical step is construct the input $\langle M, w \rangle_n$ with length n. This is only possible since the length of w is $o(n)$. If we attempt to use the same proof for NTIME$[G(n)]$, the length of w can be $G(n)$ which can be much larger than n, and we cannot encode M and w both in the n-bit string.

2.2 Part (b)

The original proof would just work, the only difference is to observe that the new language H is indeed also in NTIMEGUESS$[T(n), n]$, since the simulation of a non-deterministic machine with $n/10$ witnesses itself can be done with n witnesses.

3 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 2 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!

4 Solution to Problem 2

Let n_1 be a sufficiently large integer. For every $i \in \mathbb{N}$, we set $n_i = T(n_{i-1})^{1.1} + 1$.

Given the description of a non-deterministic machine M (for a description of an normal nondeterministic TM, we add a timer to make it stop in $T(n)^{1.1}$ time), we can construct a language $L \in \text{NTIME}[T(n)^2]$ such that if M runs in $\text{NTIME}[T(n)]$ time, for every sufficiently large $i \in \mathbb{N}$, there exists an $n \in [n_i, T(n_i)^{1.1}]$ such that $L_n \neq M_n$.

We will define our H as follows. First, for the description of a NTM and a witness $w \in \{0,1\}^*$, and an integer $n \in \mathbb{N}$. We define $\langle M, w\rangle_n$ to be an encoding of the pair $\langle M\rangle$ (the description of M) and w (the witness), such that $\langle M, w\rangle_n \in \{0,1\}^n$. Note that if $|\langle M\rangle| + |w| > n/2$, we simply set $\langle M, w\rangle_n = 0^n$ (i.e., we give up encoding such pairs).

First, for every $n \in [n_i, T(n_i)^{1.1})$, we set (below $\langle M, 0\rangle_n$ denotes an encoding of the pair $\langle M\rangle$ and length-1 witness 0)

$$H(\langle M, 0\rangle_n) = M(\langle M, 0\rangle_{n+1}).$$

Note that M runs in $O(T(n))$ non-deterministic time, so H above takes $O(T(n+1)) \leq T(n)^2$ time, since T is a polynomial.

Now, let $m = c \cdot T(n) < T(n)^{1.1}$ be the running time of M on input length n.

Let $w_1, \ldots, w_{2^m}$ be a sequence of all possible strings of length m.

Let $r = T(n)^{1.1}$ be the right side of the interval $[n_i, T(n_i)^{1.1}]$.

We then define

$$H(\langle M, 0\rangle_r) = M(\langle M, w_1\rangle_r).$$

and

$$H(x) = \begin{cases} M(\langle M, w_{i+1}\rangle_r) \wedge [D_M(\langle M, 0\rangle_n, w_i) = 0] & \text{if } x = \langle M, w_i\rangle_r \text{ for some } i \in [1, 2^m) \\ [D_M(\langle M, 0\rangle_n, w_{2^m}) = 0] & \text{if } x = \langle M, w_{2^m}\rangle_r \\ 0 & \text{otherwise} \end{cases}$$

For H on other values not considered above, we simply set $H(x) = 0$.

Note that $H \in \text{NTIME}[T(n)^2]$.

Now, for the sake of contradiction, suppose M computes H on all n-bit inputs for $n \in [n_i, T(n_i)^{1.1}]$.

Then we know That

$$H(\langle M, 0\rangle_n) = H(\langle M, 0\rangle_{n+1})$$

for all $n \in [n_i, T(n_i)^{1.1})$ and

$$H(\langle M, 0\rangle_r) = H(\langle M, w_1\rangle_r).$$

THis gives us $H(\langle M, 0\rangle_n) = H(\langle M, w_1\rangle_r)$.

Then, by the definition of H on r-bit inputs, we also have

$$H(\langle M, w_1\rangle_r) = \bigwedge_{i=1}^{2^m} [D_M(\langle M, 0\rangle_n, w_i) = 0]$$

Reading the right side above, it is true if and only if $D_M(\langle M, 0\rangle_n, w_i) = 0$ for all $i \in [1, 2^m]$, which means M rejects the input $\langle M, 0\rangle_n$.

Therefore, we have $H(\langle M, w_1\rangle_n) = \neg M(\langle M, 0\rangle_n)$, which contradicts the assumption that M computes H on all n-bit inputs for $n \in [n_i, T(n_i)^{1.1}]$.

5 Problem 3: Refuter for Theorem 1

Theorem 1 implies that, for the corresponding hard language $L \in \text{NTIME}[T(n)]$, for every $\text{NTIMEGUESS}[G(n), W(n)]$ 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 $\text{NTIMEGUESS}[G(n), W(n)]$ 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 $\mathcal{O} \in \text{NP}$, 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.

6 Solution to Problem 3

6.1 Part (a)

Note that we can maintain two points ℓ, r such that $a_\ell \neq a_r$ and $\ell < r$. They are initialized to 1 and N. Each time we query $m = \lfloor (\ell + r)/2 \rfloor$, we can compare a_m with a_ℓ and a_r. If $a_m \neq a_\ell$, we set $r = m$, if $a_m \neq a_r$, we set $\ell = m$. We stop if $\ell + 1 = r$.

6.2 Part (b)

Recall the definition of $H \in \text{NTIME}[T(n)]$ as

$$H(x) = \begin{cases} M(\langle M, w_{i+1} \rangle_n) \wedge [D_M(\langle M, w_1 \rangle_n, w_i) = 0] & \text{if } x = \langle M, w_i \rangle_n \text{ for some } i \in [1, 2^{n/10}) \\ [D_M(\langle M, w_1 \rangle_n, w_{2^{n/10}}) = 0] & \text{if } x = \langle M, w_{2^{n/10}} \rangle_n \\ 0 & \text{otherwise} \end{cases}$$

Now, there are two cases, the first case is that $M(\langle M, w_1 \rangle_n) = 0$ (we can query the NP oracle to check this condition). In particular, this means $D_M(\langle M, w_1 \rangle_n, w_i) = 0$ for all $i \in [1, 2^{n/10}]$. Then we have

$$H(x) = \begin{cases} M(\langle M, w_{i+1} \rangle_n) & \text{if } x = \langle M, w_i \rangle_n \text{ for some } i \in [1, 2^{n/10}) \\ 1 & \text{if } x = \langle M, w_{2^{n/10}} \rangle_n \end{cases}$$

Now, consider the sequence

$$M(\langle M, w_1 \rangle_n), M(\langle M, w_2 \rangle_n), \ldots, M(\langle M, w_{2^{n/10}} \rangle_n), H(\langle M, w_{2^{n/10}} \rangle_n)$$

From the discussions above, we know that $0 = M(\langle M, w_1 \rangle_n) \neq H(\langle M, w_{2^{n/10}} \rangle_n) = 1$. Then, by part (a), we can find an index i such that either $M(\langle M, w_i \rangle_n) \neq M(\langle M, w_{i+1} \rangle_n)$ or $H(\langle M, w_{2^{n/10}} \rangle_n) \neq M(\langle M, w_{2^{n/10}} \rangle_n)$. Since $M(\langle M, w_{i+1} \rangle_n) = H(\langle M, w_i \rangle_n)$, we know that $M(\langle M, w_i \rangle_n) \neq H(\langle M, w_i \rangle_n)$ as well. This solves the problem when $M(\langle M, w_1 \rangle_n) = 0$.

The second case is that $M(\langle M, w_1 \rangle_n) = 1$. In particular, this means $D_M(\langle M, w_1 \rangle_n, w_i) = 1$ for some $i \in [1, 2^{n/10}]$.

Note that by a binary search, we can find the first j such that $D_M(\langle M, w_1 \rangle_n, w_j) = 1$. In particular, this also means That

$$H(x) = \begin{cases} M(\langle M, w_{i+1} \rangle_n) & \text{if } x = \langle M, w_i \rangle_n \text{ for some } i \in [1, j) \\ 0 & \text{if } x = \langle M, w_j \rangle_n \end{cases}$$

Now, consider the sequence

$$M(\langle M, w_1 \rangle_n), M(\langle M, w_2 \rangle_n), \ldots, M(\langle M, w_j \rangle_n), H(\langle M, w_j \rangle_n)$$

We can apply the part (a) and argue similarly to solve the problem.

7 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 $r \in \{0,1\}^{p(|x|)}$.

8 Solution to Problem 4

8.1 Part (a)

Essentially just take a hard enough oracle would suffice. One can take the PSPACE-complete language TQBF as the oracle. This would collapse both $\mathrm{P}^{\mathcal{O}_1}$ and $\mathrm{BPP}^{\mathcal{O}_1}$ to PSPACE.

8.2 Part (b)

For the oracle $\mathcal{O}_2$, we consider the following language:

$$L = \{1^n : \text{there are at least } 2/3 \text{ fractions of } n\text{-bit strings } x \text{ such that } \mathcal{O}_2(x) = 1 \}.$$

Furthermore, we will promise that for our oracle $\mathcal{O}_2$, either there are at least $2/3$ fractions of n-bit strings x such that $\mathcal{O}_2(x) = 1$, or there are at most 0 fractions of n-bit strings x such that $\mathcal{O}_2(x) = 1$.

Let n_1 be a sufficiently large integer. For every $i \in \mathbb{N}$, we set $n_i = 2^{n_{i-1}}$. This is to ensure that for all $n^{\log n}$ machine on input length n_i, they cannot query input length n_{i+1} on $\mathcal{O}_2$.

Now, on input length n_i we will make sure the i-th $n^{\log n}$-time TM M_i cannot compute L on input length n_i.

To do this, we simply simulate M_i with the current $\mathcal{O}_2$ (all unset values are 0) on input 1^{n_i} for $n_i^{\log n_i}$ steps. If M_i outputs 1, we set $\mathcal{O}_2$ to be all zero on n_i-bit inputs. Otherwise if M_i outputs 0, we set $\mathcal{O}_2$ to be all-one except for the queried inputs. This would ensure that M_i cannot compute L on input length n_i.

8.3 Part (c)

First we can set the oracle $\mathcal{O}$ to be TQBF, this would collapse $\mathrm{P}^{\mathcal{O}}$ and $\mathrm{BPP}^{\mathcal{O}}$ to PSPACE.

Next, we are going to construct another oracle $\mathcal{O}'$ such that $\mathrm{P}^{\mathcal{O},\mathcal{O}'} = \mathrm{BPP}^{\mathcal{O},\mathcal{O}'}$, yet $\mathrm{P}^{\mathcal{O},\mathcal{O}'} \neq \mathrm{NP}^{\mathcal{O},\mathcal{O}'}$. Then our final oracle $\mathcal{O}_3$ would just be putting these two oracle $\mathcal{O}$ and $\mathcal{O}'$ together.

Let n_1 be a sufficiently large integer. For every $i \in \mathbb{N}$, we set $n_i = 2^{n_{i-1}}$.

For the oracle $\mathcal{O}'$, we consider the following language:

$$L = \{1^n : \text{there is a } n\text{-bit string } x \text{ such that } \mathcal{O}'(x) = 1 \}.$$

Now, on input length n_i we will make sure the i-th $n^{\log n}$-time TM M_i cannot compute L on input length n_i.

To do this, we simply simulate M_i with the current $\mathcal{O}'$ (all unset values are 0) on input 1^{n_i} for $n_i^{\log n_i}$ steps. If M_i outputs 1, we set $\mathcal{O}'$ to be all zero on n_i-bit inputs. Otherwise if M_i outputs 0, we pick a entry $x \in \{0,1\}^{n_i}$ that is not queried yet, and set $\mathcal{O}'(x) = 1$, and set everything else to 0. This would ensure that M_i cannot compute L on input length n_i.

The point is that, we can pick x in a way that it is queried by the first n_i $n^{\log n}$-time randomized TM with probability at most $2^{-n_i/2}$, since there are 2^{n_i} possible strings. One can argue that this probability is too small and we still have $\mathrm{P}^{\mathcal{O},\mathcal{O}'} = \mathrm{BPP}^{\mathcal{O},\mathcal{O}'}$.