CSC463H1: Computational Complexity Theory

Assignment 3 Solutions

Alexander Meng
mengale1

November 8, 2025

Question 1 (10 points)

Show that the set of incompressible strings contains no infinite subset that is Turing recognizable.

Definition 1. A string $x \in \{0,1\}^*$ is *incompressible* if its Kolmogorov complexity satisfies $K(x) \geq |x|$, where $K(x)$ is the length of the shortest program that outputs x.

Proof

We prove this by contradiction. Suppose there exists an infinite subset S of incompressible strings that is Turing recognizable. Let M be a Turing machine that recognizes S.

Step 1: Enumeration of S.
Since M recognizes S, we can effectively enumerate the elements of S by dovetailing: systematically run M on all inputs in lexicographic order, and whenever M accepts a string x, add x to our enumeration. This gives us a sequence $s_1, s_2, s_3, \ldots$ of all strings in S.

Step 2: Construction of a compression procedure.
Consider the following compression scheme for strings in S:

- To describe a string $s_i \in S$ (the i-th element in the enumeration), we need only:
  (a) A description of the Turing machine M (fixed constant c_M)
  (b) The index i in binary (requires $\lceil \log_2 i \rceil$ bits)
  (c) A program to enumerate S using M and extract the i-th element (fixed constant c_P)

Therefore, for the i-th string $s_i \in S$:

$$K(s_i) \leq c_M + \lceil \log_2 i \rceil + c_P = O(\log i)$$

Step 3: Deriving the contradiction.
Since S is infinite, it contains arbitrarily long strings. For any length n, there exists some index j such that $|s_j| \geq n$.
However, by our assumption, every string in S is incompressible, so:

$$K(s_j) \geq |s_j| \geq n$$

But from Step 2, we have:

$$K(s_j) = O(\log j)$$

For sufficiently large j, we have $\log j < |s_j|$ (since strings in S can be arbitrarily long while their indices grow only logarithmically in description). This contradicts the incompressibility of s_j.

Conclusion.
The assumption that an infinite subset of incompressible strings is Turing recognizable leads to a contradiction. Therefore, no infinite subset of incompressible strings is Turing recognizable. $\square$

Question 2 (10 points)

Let *MODEXP* be the set of all tuples (a, b, c, p) such that a, b, c, p are integers in binary, and $a^b \equiv c \pmod p$. Show that *MODEXP* $\in P$.

Definition 2. The language *MODEXP* is defined as:

$$MODEXP = \{(a, b, c, p) : a, b, c, p \in \mathbb{Z},\ a^b \equiv c \pmod p\}$$

where all integers are represented in binary.

Proof

To show $MODEXP \in P$, we construct a deterministic polynomial-time algorithm that decides membership in *MODEXP*.

Algorithm: Fast Modular Exponentiation
Given input (a, b, c, p) in binary:

1. **Input validation:** Check that $p > 0$. If $p = 0$, reject (division by zero undefined). If $p = 1$, accept if $c \equiv 0 \pmod 1$ (always true).
2. **Compute $a^b \bmod p$ using binary exponentiation:**
   - Initialize: result $\leftarrow 1$, base $\leftarrow a \bmod p$
   - Write b in binary: $b = \sum_{i=0}^{k-1} b_i 2^i$ where $k = \lceil \log_2(b+1) \rceil$
   - For $i = 0$ to $k - 1$:
     - If $b_i = 1$: result $\leftarrow$ (result $\times$ base) mod p
     - base $\leftarrow$ (base $\times$ base) mod p
3. **Compare:** Accept if result $\equiv c \pmod p$; otherwise reject.

Correctness:
The algorithm correctly computes $a^b \bmod p$ by the binary representation of b:

$$a^b = a^{\sum_{i=0}^{k-1} b_i 2^i} = \prod_{i=0}^{k-1} (a^{2^i})^{b_i}$$

Each iteration maintains the invariant that base $= a^{2^i} \bmod p$ and result accumulates the product of relevant powers.

Time Complexity Analysis:
Let n be the total input size in bits. Then:

- $|a|, |b|, |c|, |p| = O(n)$ bits each
- Number of iterations: $k = O(\log b) = O(n)$ (since b has at most n bits)
- Each iteration performs:
  - At most 2 multiplications of $O(n)$-bit numbers: $O(n^2)$ time using standard multiplication

- Modulo operation: $O(n^2)$ time using standard division

- Total time: $O(n) \times O(n^2) = O(n^3)$

Since the algorithm runs in time polynomial in the input size and always halts, we have $MODEXP \in P$. $\square$

Question 3 (10 points)

Show that if $P = NP$, then every language $A \in P$ except $A = \varnothing$ and $A = \Sigma^*$ is NP-complete.

Definition 3. A language A is NP*-complete* if:

(i) $A \in NP$, and

(ii) For every language $L \in NP$, we have $L \leq_P A$ (i.e., L polynomial-time reduces to A).

Proof

Assume $P = NP$. Let $A \in P$ be any language such that $A \neq \varnothing$ and $A \neq \Sigma^*$.

Step 1: Show $A \in NP$.
Since $A \in P$ and $P = NP$ by assumption, we immediately have $A \in NP$. ✓

Step 2: Show every $L \in NP$ reduces to A in polynomial time.
Let $L \in NP$ be arbitrary. Since $P = NP$, we have $L \in P$. Therefore, there exists a polynomial-time Turing machine M_L that decides L.
Since A is nontrivial:

- There exists some string $y_{\text{yes}} \in A$ (since $A \neq \varnothing$)
- There exists some string $y_{\text{no}} \notin A$ (since $A \neq \Sigma^*$)

Both y_{yes} and y_{no} are fixed strings that can be hardcoded into our reduction.

Construction of the reduction $f : \Sigma^* \to \Sigma^*$:
Define f as follows on input x:

1. Run M_L on input x for at most $p(|x|)$ steps, where p is the polynomial bound on M_L's running time.
2. If M_L accepts x, output $f(x) = y_{\text{yes}}$.
3. If M_L rejects x, output $f(x) = y_{\text{no}}$.

Verification:

- **f is computable in polynomial time:**

  The reduction runs M_L for polynomial time $p(|x|)$, then outputs a fixed string. Total time is $O(p(|x|)) = \text{poly}(|x|)$. ✓

- **f is a valid reduction:**

  We need to show $x \in L \iff f(x) \in A$.

$$\begin{aligned} x \in L &\iff M_L \text{ accepts } x \\ &\iff f(x) = y_{\text{yes}} \\ &\iff f(x) \in A \end{aligned}$$

  Therefore, $L \leq_P A$. ✓

Conclusion.
Since $A \in NP$ and every language $L \in NP$ reduces to A in polynomial time, A is NP-complete. This holds for all nontrivial languages $A \in P$ under the assumption $P = NP$. $\square$

Remark. This result shows that if $P = NP$, the notion of NP-completeness becomes trivial: nearly every language in P would be NP-complete. This is one reason why $P \neq NP$ is widely believed to be true.

Question 4 (10 points)

Let *INT-FACT* be the set of all pairs of binary integers (m, n) such that m has a factor less than n and greater than one. Show that *INT-FACT* is in NP and in co-NP. Show that if *INT-FACT* $\in P$, then there is a polynomial-time algorithm for factoring binary integers.

Definition 4. The language *INT-FACT* is defined as:

$$\textit{INT-FACT} = \{(m, n) : m, n \in \mathbb{Z}^+,\ \exists d \in \mathbb{Z} \text{ such that } 1 < d < n \text{ and } d \mid m\}$$

where all integers are represented in binary.

Part (a): *INT-FACT* $\in$ NP $\cap$ co-NP

Proof that *INT-FACT* $\in$ NP

To show $\textit{INT-FACT} \in NP$, we construct a polynomial-time verifier.

Certificate: A divisor d such that $1 < d < n$ and $d \mid m$.

Verifier V: On input $((m, n), d)$:

1. Check that $1 < d < n$. If not, reject.
2. Compute $m \bmod d$ using polynomial-time division.
3. Accept if $m \bmod d = 0$ (i.e., $d \mid m$); otherwise reject.

Correctness:

- If $(m, n) \in \textit{INT-FACT}$, then there exists a divisor d with $1 < d < n$ and $d \mid m$. The verifier accepts with certificate d.
- If $(m, n) \notin \textit{INT-FACT}$, no such d exists, so no certificate will cause the verifier to accept.

Time Complexity: The verifier runs in time polynomial in $|m|+|n|+|d| = O(\log m + \log n)$, since:

- Comparison operations: $O(\log n)$
- Division $m \bmod d$: $O(\log^2 m)$ using standard algorithms

Therefore, $\textit{INT-FACT} \in NP$. □

Proof that *INT-FACT* $\in$ co-NP

To show $\textit{INT-FACT} \in$ co-NP, we show that $\overline{\textit{INT-FACT}} \in NP$.

$$\overline{\textit{INT-FACT}} = \{(m, n) : \forall d,\ 1 < d < n \implies d \nmid m\}$$

This states: "m has no divisor in the range $(1, n)$", which means either m is prime and $n > m$, or all of m's nontrivial divisors are $\geq n$.

Certificate Strategy:
We use the fact that primality testing is in P. This is a fundamental result in complexity theory that allows us to verify primality in polynomial time.
For $(m, n) \in \overline{INT\text{-}FACT}$:

- **Case 1:** $m = 1$. Then m has no divisors > 1, so $(m, n) \in \overline{INT\text{-}FACT}$ for any $n \geq 2$.
- **Case 2:** m is prime and $n > m$. Certificate: primality proof for m.
- **Case 3:** m is composite with smallest prime factor $p \geq n$. Certificate: the complete prime factorization of m.

Verifier V': On input $((m, n), \pi)$ where π is a certificate:

1. If $m = 1$ and $n \geq 2$, accept.
2. If π claims "m is prime":
   - Verify m is prime using polynomial-time primality testing (PRIMES $\in P$).
   - If m is prime and $n > m$, accept; otherwise reject.
3. If π provides a factorization $m = p_1^{e_1} \cdots p_k^{e_k}$:
   - Verify each p_i is prime using polynomial-time primality testing.
   - Verify $\prod_{i=1}^{k} p_i^{e_i} = m$ using polynomial-time multiplication.
   - Check that $\min(p_1, \ldots, p_k) \geq n$.
   - If all checks pass, accept; otherwise reject.

Time Complexity:

- Primality verification: polynomial time (using PRIMES $\in P$)
- Verification of factorization: polynomial in the size of the factorization
- Total: polynomial time

Certificate Size: The factorization certificate $m = p_1^{e_1} \cdots p_k^{e_k}$ has polynomial size because:

- Number of distinct prime factors: $k \leq \log_2 m$ (since each prime ≥ 2)
- Each prime p_i requires $O(\log m)$ bits to represent
- Each exponent $e_i \leq \log_2 m$ requires $O(\log \log m)$ bits
- Total certificate size: $O(k \cdot \log m) = O(\log^2 m)$ bits = polynomial

Therefore, $\overline{INT\text{-}FACT} \in NP$, which implies $INT\text{-}FACT \in$ co-NP. $\square$

Part (b): If *INT-FACT* $\in P$, then factoring is in P

Proof

Assume *INT-FACT* $\in P$. We construct a polynomial-time algorithm to factor any integer m.

Factoring Algorithm:
Input: Integer $m > 1$ in binary.
Output: A nontrivial factor of m, or "prime" if m is prime.

1. **Check if m is prime:** Run $(m, m) \in \textit{INT-FACT}$?
   - If $(m, m) \notin \textit{INT-FACT}$, then m has no divisor in $(1, m)$, so m is prime. Output "prime" and halt.
   - Otherwise, m is composite. Proceed to Step 2.
2. **Binary search for smallest nontrivial divisor:**
   Let $\ell = 2$ and $r = m$. We perform binary search to find the smallest divisor of m.
   - While $\ell < r$:
     (a) Set $\text{mid} = \lfloor (\ell + r)/2 \rfloor$.
     (b) Query: Is $(m, \text{mid} + 1) \in \textit{INT-FACT}$?
     (c) If yes: m has a divisor in $(1, \text{mid} + 1)$, so the smallest divisor is $\leq \text{mid}$. Set $r = \text{mid}$.
     (d) If no: m has no divisor in $(1, \text{mid} + 1)$, so the smallest divisor is $> \text{mid}$. Set $\ell = \text{mid} + 1$.
   - When the loop terminates, $\ell = r$ is our candidate for the smallest divisor.
   - **Verify divisibility:** Compute $m \bmod \ell$ using polynomial-time division.
   - If $\ell \mid m$ (i.e., $m \bmod \ell = 0$), then ℓ is the smallest nontrivial divisor of m.
3. **Output:** Return ℓ as a nontrivial factor of m.

Correctness:
The binary search maintains the following invariant:

- All nontrivial divisors of m (if any exist) are $\geq \ell$
- If m is composite, there exists a divisor $\leq r$

When the loop terminates with $\ell = r$, we have identified a single candidate value. The divisibility check $m \bmod \ell = 0$ confirms that ℓ is indeed a divisor. Since all smaller values have been eliminated by the binary search, ℓ must be the smallest nontrivial divisor.

Time Complexity:

- Binary search performs $O(\log m)$ iterations.
- Each iteration queries *INT-FACT*, which takes polynomial time by assumption.
- Total time: $O(\log m) \times \text{poly}(\log m) = \text{poly}(\log m)$.

Since the algorithm runs in polynomial time and correctly finds a nontrivial factor of m, we have shown that if *INT-FACT* $\in P$, then integer factorization is in P. □

Remark. This result shows that solving the decision problem *INT-FACT* efficiently is at least as hard as factoring integers, a problem not known to be in P but also not proven to be outside P. The containment of *INT-FACT* in both NP and co-NP suggests it may not be NP-complete (under the assumption $NP \neq$ co-NP).

— End of Solutions —