CS Theory (Fall '25) Assigned: Oct 7, 2025

Homework 3

Instructors: *William Pires and Toniann Pitassi* Due: Oct 16, 2025 at 11am

Collaboration: Collaboration with other students in the class homework is allowed. However, you must write up solutions by yourself and understand everything that you hand in. **You may not use the web or AI tools when working on homework questions, and you are required to list who you collaborated with on each problem, including any TAs or students you discussed the problems with in office hours. List any reference materials consulted other than the lectures and textbook for our class. See the course webpage for more details.**

Formatting: Write the solution to each part of each problem on a separate page. Be sure to correctly indicate which page each problem appears on in GradeScope. Please do not write your name on any page of the submission; we are using anonymous grading in GradeScope. If we can't find your solution to any problems because it was not properly tagged with the page, or if handwriting is not legible, you will receive 0 percent on these problems.

Grading: There are 60 total possible points. For all problems you have the option of answering "Don't know" on any parts of the question, and you will receive 20 percent of the total marks for those parts. *You will also be graded on clarity and brevity of your answers. For each question below, we have added guidelines for the length of your solution.*

0 Exercises

Here is a recommended exercise. This will not be graded, so you should not turn in your solution.

(a) Give a streaming algorithm $\mathcal{A}$ (give the 5-tuple) for recognizing the following language L using $O(\log(n))$ space. State the space usage of your algorithm in big-O notation, as a function of n, the length (number of digits) of the input. $L = \{x \in \{0,1\}^* \mid |x| \text{ is a power of 2 }\}$.

Consider the following algorithm: the input alphabet is $\Sigma = \{0,1\}$ and the memory alphabet is $\Pi = \{0,1\}$. The content of memory M will be the binary representation of the number of symbols (0 or 1) seen so far, and at the end we output 1 or 0 based on whether the number of symbols seen is a power of 2. We do so with the following rules:

1. Initialization rule I: $M = 0$, which is the binary representation of 0.
2. Update rule δ: on input $\sigma \in \{0,1\}$, update M to be $\langle \text{num}(M) + 1 \rangle$. This increments M by 1 in binary.
3. Stopping rule γ: if M is a power of 2, output 1. Otherwise output 0.

The memory M keeps track of the number of symbols that have been streamed so far, so at the end of the algorithm, $M = \langle |x| \rangle$, the length of the input in binary. Thus the algorithm outputs 1 on exactly the inputs x where $|x|$ is a power of 2.

Since the memory records in binary how many symbols have been read, and at most $|x| = n$ symbols will be read, the memory will use at most $\lceil \log_2 n \rceil = O(\log n)$ bits.

1 Asymptotic Analysis (10 points)

For each of the following answer T (true) or F (false). No explanation necesssary. [1] Whenever the base of a log isn't specified, we mean $\log_2$.

(a) $2^n = O(n^n)$.

(b) $\log_e(n) = O(\log_{100}(n))$.

(c) $n^2 \log\log(n) = \Omega(n \log(n))$.

(d) $n^3 = n^{\Omega(5)}$.

(e) $\log_2(n) = \Omega(\log_2(n^{100}))$.

(f) $n^{1/\log n} = \text{poly}(n)$.

(g) $16^{\log_2(n)} + 20n^2 + \sqrt{n} + 45 = \text{poly}(n)$.

(h) $\frac{n}{\log\log\log n} = o(n)$.

(i) $100^{-100} = o(1)$.

(j) For all functions $f, g : \mathbb{N} \to \mathbb{R}^+$, if $f = o(g(n))$, then $\frac{1}{2}(f(n) + g(n)) = o(g(n))$.

(a) True. For n larger than 2, $2^n < n^n$.

(b) True. $\log_e(n) = \log(n)/\log(e)$, so $\log_e(n) = \log(e)\log_{100}(n)$.

(c) True. For all $n > 4$, $\log\log(n) \geq 1$, so $n^2 \log\log n \geq n^2$. For all $n > 1$, $n \log n < n^2$. So, $n \log n < n^2 \leq n^2 \log\log n$ for $n > 4$.

(d) True. $3 = \Omega(5)$.

(e) True. $\log_2(n^{100}) = 100\log_2(n)$, so $\log_2(n) \geq (1/100)\log_2(n^{100})$ for all n.

(f) True. For $n > 2$, $\log n \geq 1$, so $1/\log n \leq 1$. This means for $n > 2$, $n^{1/\log n} \leq n^1$.

(g) True. We can go term-by-term. $16^{\log_2(n)} = n^4$, which is clearly in $\text{poly}(n)$. $20n^2 \in \text{poly}(n)$. $\sqrt{n} = n^{1/2} \in \text{poly}(n)$. $45 \in \text{poly}(n)$. Since all four terms are in $\text{poly}(n)$, the whole expression is (check for yourself why $\text{poly}(n)$ is closed under addition!).

(h) True. For $c > 0$, let $n_0 = 2^{2^{2^{1/c}}}$. Then, for $n \geq n_0$, $\log\log\log n \geq 1/c$. So for these n, $\frac{n}{\log\log\log n} \leq cn$, as desired.

(i) False. Let $c = 100^{-100}/2$. Then $100^{-100} \not< c \cdot 1$.

(j) False. Let $f(n) = 0$ and $g(n) = n$. Clearly, $f(n) = o(g(n))$. However, $\frac{1}{2}(f(n) + g(n)) = \frac{1}{2}n$. This is not $o(g(n))$.

[1] We will not grade explanations but it is important that you can explain your answer so we will be providing explanations/proofs in the solutons.

2 Nonregular Languages (10 points)

Let FlipFlop $\subseteq \{0,1\}^*$ be the following language: $w \in \{0,1\}^*$ is in FlipFlop if and only if w can be written as $u \cdot \overline{u}$ where $\overline{u}$ is the string obtained by replacing all 0's in u by 1's, and replacing all 1's in u by 0's. For example, $w = 0100010111$ is in FlipFlop since $w = u\overline{u}$ where $u = 01000$ and $\overline{u} = 10111$.

Prove that FlipFlop is not regular by giving a $\Omega(n)$ one-way communication lower bound for the associated communication problem. Do not use the Pumping Lemma. However you may use anything from the course notes on streaming/communication complexity. Your answer should be half a page or less.

Solution using one way communication protocol:

Proof. Let us assume for the sake of contradiction there exists a one-way protocol for FlipFlop that, on inputs of length $2n$, has communication complexity $s(2n) < n$, meaning $s(2n) \leq n-1$. Then the number of possible messages Alice can send is $\sum_{i=0}^{s(2n)} 2^i = 2^{s(2n)+1} - 1 < 2^n$. However, there are 2^n possible strings of length n that can be Alice's input. By pigeonhole principle, there must exist two distinct strings u, v of length n which will cause Alice to send the same message; let's call this message m^*. Now consider the protocol on two different inputs: $w = u \circ \overline{u}$ and $w' = v \circ \overline{u}$. Note that w is in FlipFlop, but w' is not in FlipFlop, since as argued above $u \neq v$. On both of these inputs Alice sends the same message m^* so Bob's output, $g(m^*, \overline{u})$, must be the same on both w and w'. (Since his output depends only on m^* and $\overline{u}$.) Thus we reach a contradiction since on w a correct protocol should output 1 and on w' a correct protocol should output 0, but Bob will give the same answer on both inputs.

□

Solution using communication complexity matrix:

Proof. Consider the communication complexity matrix where we label the rows by the length-n strings in lexicographic order, and for columns we follow the same lexicographic order but label the column by the complement of the string. That is, if row i is labeled string u then column i is labeled $\overline{u}$. Let the entry at row i (labeled by string u) and column j (labeled by $\overline{v}$) be 1 if $u \cdot \overline{v} \in$ FlipFlop and 0 otherwise.

We observe that this matrix is just like the matrix for EQ (equality language from class). For row i and column i, the corresponding string is $u \cdot \overline{u} \in$ FlipFlop because the string at column i is the complement of row i. On the other hand, for row i and column $j \neq i$, the corresponding string is $u \cdot \overline{v} \notin$ FlipFlop because $j \neq i$ implies $u \neq v$.

So for any length-n strings $u \neq v$, the strings $u\overline{u}$ and $v\overline{v}$ must be in different partitions (that is, Alice sends different messages if she receives u versus v). There are 2^n different length-n strings and thus 2^n different partitions, each corresponding to a different message. Therefore Alice needs at least $n-1$ bits to send her message, so the one way communication protocol's complexity has lower bound $\Omega(n)$. □

3 Streaming Algorithms (20 points)

Let Sum $= \{w \in \{-1,1,0\}^* \mid$ the digits in w sum to $0\}$.

(a) Give a streaming algorithm $\mathcal{A}$ (give the 5-tuple) for recognizing Sum with space complexity $O(\log n)$ where n is the input length. Give a brief explanation of your algorithm (1-2 sentences) and a brief justification of its space complexity (1 sentence). [2]

[2] Your explanation should be here to help us understand your algorithm and what you're keeping track of.

(b) Prove that this language is not regular either by a direct argument (for example, the direct proof of Theorem 2 from Oct 6/7 notes) or by proving that it does not have constant one-way communication complexity. Do not use the Pumping Lemma. However, you may use anything from the course notes on streaming/communication complexity. Your answer should be less than one page.

(a) $\mathcal{A} = (\Sigma, \Pi, \mathcal{I}, \delta, \gamma)$ where

$$\Sigma = \{-1, 1, 0\}$$
$$\Pi = \{-1, 1, 0\}$$
$$\mathcal{I} = 0$$
$$\delta(M, \sigma) = \langle num(M) + \sigma \rangle$$
$$\gamma(M) = \begin{cases} 1 & num(M) = 0 \\ 0 & \text{otherwise.} \end{cases}$$

Explanation: This algorithm keeps track of the sum of the digits seen so far, stores the sum in M, and at the end checks whether M is 0 or not. Please see the course lecture notes for details on how to store an integer in binary representation.

Space complexity: The largest possible sum for an input of length n is n (or $-n$). We can represent this using $O(\log n)$ bits (plus potentially a negative sign), so $|M| = O(\log n)$.

(b) Solution 1 (direct argument):

Suppose for the sake of contradiction that there is a DFA $(Q, \Sigma, q_0, F, \delta)$ that recognizes Sum. Let $|Q| = n$. Consider a string $u \in \Sigma^n$. There are $2n+1$ possible values for the sum of the digits in u (the minimum is $-n$ and the maximum is n. Since there are only n states, by the pigeonhole principle, there must be two inputs u, u' such that (1) the sum of the digits in u is different from the sum of the digits in u' and (2) $\delta(q_0, u) = \delta(q_0, u')$. Let v be a string whose digits sum to the negative of the sum of the digits in u. Then $\delta(q_0, uv) \in F$, because the digits in uv will sum to 0. However, since $\delta(q_0, u) = \delta(q_0, u')$, we know that $\delta(q_0, u'v) = \delta(q_0, uv) \in F$. But the sum of the digits in v cannot be both the negative of the sum of the digits in u and the negative of the sum of the digits in u' (because they are different sums). Therefore, the sum of the digits in $u'v$ is *not* 0, and the DFA does not correctly reject $u'v$.

Solution 2 (one-way communication complexity):

Suppose for the sake of contradiction that there is a one-way communication protocol for Sum that on inputs of length $2n$, has communication complexity $s(2n) \leq \log_2(n) - 1$. Consider an input uv where $|u| = |v| = n$. There are $2n + 1$ possible values for the sum of the digits in u (the minimum is $-n$ and the maximum is n. Since the numbers of bit sent by Alice is $\log_2(n) - 1$, there at most $\sum_{i=0}^{s(2n)} 2^i = 2^{s(2n)+1} - 1 \leq 2^{\log_2(n)} - 1 \leq n - 1$ possible messages that Alice can send. Therefore, by the pigeonhole principle, there must be two inputs u, u' such that (1) the sum of the digits in u is different from the sum of the digits in u' and (2) Alice sends the same message α on u and u'. Let v be the string obtained by flipping all digits in u to their negative. In particular we have that the sum of the digits in v is the negative of the sum of the digits in u. So, we have that $uv \in$ Sum (the digits in uv must sum to 0). Therefore, the protocol needs to accept input (u, v) so $g(\alpha, v) = g(m(u), v)1$. However, we since u and u' have different sum, the sum of the digits in $u'v$ can't be 0. Hence we must have $g(\alpha, v) = g(m(u'), v) = 0$. This is a contradiction. Hence any protocol for Sum must have $s(2n) \geq \log_2(n)$, hence Sum has $\Omega(\log(n))$ communication complexity. So Sum isn't regular (Recall that if a language is regular it has $O(1)$ communication complexity).

4 Fun with Polynomials (10 points)

Let $f : \mathbb{N} \to \mathbb{R}^+$, $g : \mathbb{N} \to \mathbb{R}^+$ be functions.
Use the formal definition of poly(n) (see notes) to prove that if $f = \text{poly}(n)$ and $g = \text{poly}(n)$, then $f(g(n)) = \text{poly}(n)$. Your answer should be half a page or less.

Proof. Since $f = \text{poly}(n)$, there exists by definition constants $c_f, n_f \geq 0$ such that for all $n \geq n_f$ we have $f(n) \leq n^{c_f}$. Now let $k = \max_{n<n_f}\{f(n)\}$. Then we have

$$f(n) \leq \begin{cases} k & n < n_f \\ n^{c_f} & n \geq n_f \end{cases}$$

Therefore

$$f(g(n)) \leq \begin{cases} k & g(n) < n_f \\ g(n)^{c_f} & g(n) \geq n_f \end{cases}$$

Since g is also poly(n), there exists by definition constants $c_g, n_g \geq 0$ such that for all $n \geq n_g$ we have $g(n) \leq n^{c_g}$. So for $n \geq n_g$, we get

$$f(g(n)) \leq \begin{cases} k & g(n) < n_f \\ n^{c_g c_f} & g(n) \geq n_f \end{cases}$$

Furthermore, note that $k \leq n^{c_g c_f}$ when $n \geq k^{\frac{1}{c_g c_f}}$. We assume here that c_g and c_f are nonzero, since if $f(n) \leq n^0 = 1$ for $n \geq n_f$, then $f(n) \leq 1 \leq n^1$ for $n \geq \max\{n_f, 1\}$. The same logic applies to g.

If we let $N = \max\{k^{\frac{1}{c_g c_f}}, n_g\}$, then for all $n \geq N$, we have $f(g(n)) \leq n^{c_g c_f}$. Thus the composition of two poly-time functions is poly-time. □

5 So far away... (10 points)

In this problem we consider the following language

$$\text{Far} = \{w \in \{a, b, c\}^* \mid \text{ there are at most } \lfloor |w|/4 \rfloor \text{ symbols after the last } a \text{ in } w\}.$$

Examples:

(i) The string $w = bcbaa \in$ Far since there are 0 symbols after the last a;

(ii) $w = abcbcabb \in$ Far since there are $2 \leq \lfloor |w|/4 \rfloor$ symbols after the last a;

(iii) $w = bbbbbbcc \in$ Far since there's no a in w (so the condition to be in L is trivially true);

(iv) $w = ab \notin$ Far since there's $1 > \lfloor |w|/4 \rfloor$ symbols after the last a;

(v) $w = ccccabbb \notin$ Far since there's $3 > \lfloor |w|/4 \rfloor$ symbols after the last a.

Give a streaming algorithm $\mathcal{A}$ (give the 5-tuple) for Far of space complexity $O(\log n)$ where n is the input length. Give a brief explanation of your algorithm (2-3 sentences) and a brief justification for its space complexity (2-3 sentences). [3]

$\mathcal{A} = (\Sigma, \Pi, \mathcal{I}, \delta, \gamma)$ where

$$\Sigma = \{a, b, c\}$$
$$\Pi = \{0, 1, \perp\}$$
$$\mathcal{I} = 0 \perp \perp$$

$$\delta(M_1 \perp M_2, \sigma) = \begin{cases} \langle num(M_1) + 1 \rangle \perp \perp & \sigma \neq a, M_2 = \perp \\ \langle num(M_1) + 1 \rangle \perp \langle num(M_2) + 1 \rangle & \sigma \neq a, M_2 \neq \perp \\ \langle num(M_1) + 1 \rangle \perp 0 & \sigma = a \end{cases} \quad \text{(all numbers represented in binary)}$$

$$\gamma(M_1 \perp M_2) = \begin{cases} 1 & M_2 = \perp \\ 1 & M_2 \neq \perp \text{ and } num(M_2) \leq \lfloor num(M_1)/4 \rfloor \\ 0 & \text{otherwise} \end{cases} \quad \text{(all numbers represented in binary).}$$

Remark 1: Technically, we need $\Pi \subseteq \Sigma$, but it's ok if you omit it. We could add a, b, c these to Π and the rest of the 5-tuple wouldn't need to change.

Remark 2: you might be concerned about whether we can parse M into the form $M_1 \perp M_2$. This is clearly possible if we only used the special symbol $\perp$ as the divider, because we can just read the symbols before the divider as M_1 and the symbols after the divider as M_2. Since we reuse $\perp$ as a symbol in M_2, it's a little less clear. However, it's still fine because we read all symbols before any $\perp$'s are seen as M_1, then we read the first $\perp$ to appear as the divider, then we read all of the remaining symbols as M_2.

Explanation: This streaming algorithm keeps track of two things in memory: first, the number of symbols seen so far, and second, the number of symbols since the last a so far (with a special symbol to indicate if no a's have been seen). Every time it sees a symbol, it maintains this memory by (1) increasing the number of symbols seen so far by 1 and (2) either increasing the number of symbols since the last a by 1 (if the symbol is not a and we have seen an a before), or resetting the number of symbols since the last a to 0 (if the symbol is a). Finally, it checks to see whether the number of symbols after the last a (the second number in memory) is less than or equal to $\lfloor |w|/4 \rfloor$, using the first number in memory as $|w|$ (including the corner case where we see no a's as an accept state).

Space complexity: Consider a memory state $M_1 \perp M_2$ while reading an input of length n. The largest that M_1 or M_2 can be is n, which can be represented in binary in $O(\log n)$ bits. Since we can represent M_1 and M_2 in $O(\log n)$ bits each, the length of the memory state is at most $O(\log n) + 1 = O(\log n)$.

[3] Your explanation should be here to help us understand your algorithm and what you're keeping track of.