University of California, Los Angeles
CS 281 Computability and Complexity

Boran Erol

Midterm Exam

Problem 1

I'll call the starting point cell 0, and the point we can reach by taking k steps to the right cell k.
I've tried using the fact that cell i gets visited at least $\lfloor n/i \rfloor$ times and using the fact that the harmonic series grows logarithmically, but this fails since it over counts. I can't figure out how to get rid of the over counting and still maintain logarithmic growth. If I count cell 2^i for some i, this turns the harmonic series into a geometric series and removes the logarithmic growth.

Problem 2

We'll first prove the following useful lemma:

Lemma 1. *Let M be a single-tape Turing machine that computes some function $f : \{0,1\}^* \to \{0,1\}^*$. Assume that $C_i(x) = C_i(y)$ for some strings x, y. Then, $f(x)_j = f(y)_j$ for all $j > i$.*

Proof. We induct on the length of the crossing sequence and show that the tape content is invariant. This is clearly true for the base case when the crossing sequence is empty. Now, assume the tape contents are the same up to n crossings for some n. If n is odd, that means crossing $n + 1$ was towards the left, so M can't possibly alter the tape content to the right of the ith cell. Thus, suppose n is even (or equivalently, suppose we cross to the right in crossing $n + 1$). Then, notice that the view of the M is identical. MOre specifically, the contents of the tape cell, the state of the M are exactly the same for both x and y. Therefore, the tape content is going to remain the same since the behavior of M is fully determined by the state and the tape content. □

Let $n \in \mathbb{N}$. We'll consider inputs of the form $x0^n$ where $|x| = n$. We now prove the uniqueness of the crossing sequences.

Lemma 2. *Let M be a single-tape Turing Machine that computes $f : \{0,1\}^* \to \{0,1\}^*$ defined by $f(x) = xx$. Let x, y be unique strings of length n. Then, $C_i(x0^n) \neq C_j(y0^n)$ for all $n \leq i, j \leq 2n$.*

Proof. By contradiction, if they were equal, this would imply $x = y$ by first lemma. □

Let M be a single-tape Turing Machine that computes $f(x) = xx$. We'll prove that M runs in time $\Omega(n^2)$ for some input.
Now, for all strings x, define t_x to be length of the shortest crossing sequence in $\{C_i(x0^n) : n \leq i < 2n\}$. Let $t = \max_{x \in \{0,1\}^n} t_x$ and $\tilde{x} = \arg\max_{x \in \{0,1\}^n} t_x$.
Notice that the number of crossing sequences of length at most t can be calculated as follows:

$$1 + |Q| + |Q|^2 + ... + |Q|^t = \frac{|Q|^{t+1} - 1}{|Q| - 1}$$

Since we have 2^n distinct strings of length n, all of their shortest crossing sequences are also distinct. Then, we have

$$\frac{|Q|^{t+1} - 1}{|Q| - 1} \geq 2^n$$

Then, $t = \Omega(n)$. Then, $\tilde{x}$ has n crossing sequences, each of length $\Omega(n)$. Since every crossing sequence corresponds to at least one unique computation step of M, this implies that M runs in time $\Omega(n^2)$ when computing $\tilde{x}$.

Problem 3

Assume by contradiction there's some D that runs in time $T(n)$ and given some $\langle M_\alpha \rangle$ such that M_α halts on every input, can determine whether M_α runs in constant time.
We define $M(1^n)$ as follows:

1. Run $D(\langle M \rangle)$.
2. If D accepts, run for n steps.
3. If D rejects, halt.

Since $|\langle M \rangle|$ is constant, the first step takes constant time. If D accepts, we have a contradiction since M doesn't run in constant time. If D rejects, we again have a contradiction since M runs in constant time.

Problem 4

Let $\mathcal{L}$ be the set of languages decidable by Turing machines with a HALT oracle.

Lemma 3. *$HALT \notin \mathcal{L}$.*

Proof. Suppose by contradiction that there exists a decider D with a $HALT$ oracle for $HALT$.
Construct $M(\langle M' \rangle, x)$ as follows:

1. Run D on $(\langle M' \rangle, x)$.
2. If D accepts, run forever.
3. If D rejects, halt.

If $D(\langle M \rangle, (\langle M \rangle, \langle M \rangle))$ accepts, M runs forever on $(\langle M \rangle, \langle M \rangle)$, which is a contradiction. Similarly, if $D(\langle M \rangle, (\langle M \rangle, \langle M \rangle))$ rejects, M halts on $(\langle M \rangle, \langle M \rangle)$. Therefore, a decider D with a $HALT$ oracle for $HALT$ does not exist. $\square$

Problem 5

Let $\{f_1, f_2, ...\}$ be the sequence of all Turing-computable functions. Let M_i be the Turing Machine that computes f_i and $T_i(n)$ be the runtime of M_i on inputs of length n. Define T as follows:

$$T(n) = \max\{T_1(n), T_2(n), ..., T_n(n)\}$$

Let $c_{f_i} = \max\{T_i(1), T_i(2), ..., T_i(i)\}$. Every f_i is computable in $T_i(n) \leq T(n) + c_f$ time.
Let's now prove that no Turing Machine can compute T. Suppose by contradiction that there's some Turing Machine A that computes T. Then, we can construct a Turing Machine $M(\langle M' \rangle, x)$ that decides HALT as follows (n denotes the length of x):

1. Run $A(n)$ and set $t = A(n)$.
2. Simulate M' on x for t steps.
3. If M' halts, accept. Otherwise, reject.

Notice that M' might halt on x but not halt for another x'. In other words, M' might not be computing a computable function. However, if M' halts for x, we can construct a new Turing Machine M'' that does the exact same computation as M' on x and halts immediately for every other input. M'' clearly halts on every input and determines some Turing computable function f. Therefore, if M' halts on x, it will take at most $A(n)$ steps. Therefore, M decides HALT, which is a contradiction.

Problem 6

Here's the algorithm:

1. Calculate n^{1001} by multiplying 1 by n 1001 times.
2. Do a binary search to find $a \in \mathbb{N}$ such that $a = \sup\{b : b^{1000} \leq n^{1001}\}$.
3. Output a.

The correctness of the algorithm follows by definition. Let's now analyze the runtime of the algorithm.
Multiplying two numbers is a $O(\log n)$ operation using the second-grade algorithm. Therefore, calculating n^{1001} by multiplying 1 by n 1001 times is $O(\log n)$.
Binary search takes $O(\log n)$ iterations and every iteration takes $O(\log n)$ time since calculating b^{1000} is $O(\log n)$ and comparing two natural numbers is $O(\log n)$.
Therefore, this entire algorithm takes $O((\log n)^2)$, which is $O(n)$. Therefore, $f(n)$ is time-constructible.

Problem 7

Lemma 4. *NP is closed under intersection.*

Proof. Let L_1 and L_2 be languages in NP. Then, there exists polynomial time deterministic Turing Machines M_1, M_2 such that

$$x \in L_1 \iff \exists u_1 : M_1(x, u_1) = 1$$
$$x \in L_2 \iff \exists u_2 : M_2(x, u_2) = 1$$

and there's a constant $c > 0$ such that both M_1 and M_2 run in time n^c by taking the maximum of their respective constants.

We now define a verifier $M(x, u)$ that takes $u = u_1 || u_2$ as input. (We can tell where u_1 ends and u_2 starts by introducing a new symbol to the tape alphabet.) M then simulates $M_i(x, u_i)$ and outputs 1 if and only if both M_1 and M_2 output 1. Therefore,

$$x \in L_1 \cap L_2 \iff \exists u : M(x, u) = 1$$

M runs in polynomial time since M_1 and M_2 run in polynomial time. Thus, $L_1 \cap L_2 \in NP$. $\square$

We can adjust M in the proof above so that M accepts if and only if M_1 **or** M_2 accepts, which proves the NP is closed under finite unions.
Putting all of these together, if $L_1, L_2, L_3 \in NP$, $L_1 \cup (L_2 \cap L_3)$ is also in NP.

Problem 8

Let M be a Turing Machine that takes a tuple of inputs (x, y).

LEMMA 5. *Let $L = \{(M, x, 1^n) : M$ accepts (x, y) in time 2^{2^n} for some $y\}$. L is NP-Hard but not in NP.*

Proof. Let's first prove that L is not in NP. In order to do this, we'll prove that L is not in EXP. Assume by contradiction that there's a decider D for L that runs in exponential time. We construct the following Turing Machine N that takes as input $(\langle M\rangle, x, 1^n)$.

1. Simulate D on $(M, x, 1^n)$.
2. If D accepts, reject.
3. If D rejects, accept.

Now, consider running N on $\langle N\rangle, (\langle N\rangle, \langle N\rangle, 1^n)$. If D accepts, N will reject, which is a contradiction. If D rejects, N will accept and run in time $2^n n$, which is a contradiction. Therefore, L is not in EXP.
Let's now prove that L is NP-Hard. Let $A \in NP$ and $M(x, y)$ be the verifier for A that runs in time n^c. Notice that $x \in A \iff (M, x, 1^{\log\log(n^c)}) \in L$. Then, the function f defined by $f(x) = (M, x, 1^{\log\log(n^c)})$ produces the desired reduction. Clearly, f is polynomial time computable. ∎

L is decidable since we can just simulate M for 2^{2^n} steps for all y such that $|y| \leq 2^{2^n}$ and then accept if M accepts for some y.

Problem 9

Let ϕ be a 2-CNF with n variables and m clauses. We'll convert the CNF into a directed graph $G = (V, E)$. Let V be the set of (positive and negative) literals of ϕ. Thus, G has $2n$ vertices.
For every clause $x_i + x_j$ in ϕ (x_i, x_j could represent positive or negative literals), add two edges: (x_i', x_j) and (x_i, x_j'). Notice that this immediately implies that there's a path from x_i to x_j if and only if there's a path from x_j' to x_i'.
Now, for all $i \in \{1, 2, ..., n\}$, check if there's a path in G from x_i to x_i' and if there's a path from x_i' to x_i. If both of these paths exist, reject. Otherwise, accept.
Let's now explain what the semantics of this graph. Every clause in ϕ can be interpreted as two implications by basic rules of logic. G just captures these implications using a directed graph. In other words, $x_i \implies x_j$ is a "clause" in ϕ (which corresponds to $x_i' + x_j$) if and only if there's an edge from x_i to x_j in G.

LEMMA 6. *If there's a path from x_i to both x_j and x_j' for some j, there's a path from x_i to x_i'.*

Proof. Assume that there's a path from x_i to both x_j and x_j' for some j. Then, there's a path from x_j and x_j' to x_i'. This immediately implies the existence of two paths from x_i to x_i'. $\square$

LEMMA 7. *The algorithm correctly decides if ϕ is satisfiable.*

Proof. If the algorithm rejects ϕ, then ϕ contains a chain of clauses that correspond to the implication $x_i \implies x_i' \implies x_i$, which makes ϕ unsatisfiable.
Suppose the algorithm accepts ϕ. Here's how we construct a satisfying assignment:
Since the algorithm accepted, there's either no path from x_1 to x_1' or from x_1' to x_1. Without loss of generality, assume there's no path from x_1 to x_1'. Then, set $x_1 = 1$ and follow all outgoing edges from x_1 and set other variables values accordingly. By the lemma above, we can't set a variable to be true and false at the same time. After all variables reachable from x_1 have been set, remove the edges corresponding to those variables and repeat this process. This is guaranteed to produce a satisfying assignment. $\square$

Let's now analyze the runtime of this algorithm. The time it takes to create the graph G is $O(n)$. For every variable, we run our favorite polynomial time graph reachability algorithm twice. Since there are polynomially many variables and the composition of two polynomials is a polynomial, this is still a polynomial time operation.

Problem 10

Let $L = \{(M, x) : M \text{ runs in time } n^{\log n}\}$.
We'll prove that $L \in EXP$ but $L \notin P$, therefore proving that one of the inclusions in $P \subseteq PSPACE \subseteq EXP$ is proper.
$L \in EXP$ since we can simulate M on x using a universal Turing Machine and the simulation takes time $n^{\log n} + (\log n)^2$.
Now, by contradiction, assume $L \in P$. Then, there exists a decider D for L that runs in polynomial time. We now construct the following Turing Machine M that takes as input $(\langle M' \rangle, x)$.

1. Simulate D on $(\langle M' \rangle, x)$.
2. If D accepts, run for 2^n time.
3. If D rejects, halt.

Now, consider running M on $(\langle M \rangle, \langle M \rangle, \langle M \rangle)$. If D accepts, M runs for exponential time, which is a contradiction. Similarly, if D rejects, M runs in polynomial time (since D runs in polynomial time in assumption), which is a contradiction.

Problem 11

We're going to prove the following chain of inclusions.

$$PSPACE \subseteq P^{TQBF} \subseteq NP^{TQBF} \subseteq PSPACE$$

The first two inclusions are trivial. Now, let N be a non-deterministic Turing Machine with a $TQBF$ oracle. Given N, we can construct an M that tries every guess N makes and accepts if and only if N accepts for one guess. After every guess, M fully clears it tapes. Therefore, M never uses more space than N on a single branch of computation. Since N runs in polynomial time, it uses polynomial space for every branch. Every call N makes to $TQBF$ can also be implemented in polynomial space. Therefore, M runs in polynomial space. The fact that M and N decide the same language is immediate.

Problem 12

Lemma 8.

$$NTIME(n^{1000}) \subsetneq PSPACE$$

Proof. By the non-deterministic time hierarchy theorem, $NTIME(n^{1000}) \subsetneq NTIME(n^{1001}) \subseteq PSPACE$. ∎

Problem 13

LEMMA 9. *There's no algorithm that evaluates totally quantified Boolean formulas of size n is space $O(n^{1/1000})$.*

Proof. Assume that such an algorithm A exists.
Let L be a language such that $L \in SPACE(n^2)$ and $L \notin SPACE(n)$. Such a language exists by the space hierarchy theorem. Let M be a deterministic Turing Machine that runs in space $O(n^2)$. Then, we can create a TQBF ϕ of n^4 variables such that $\phi(x)$ evaluates to true if and only if $M(x)$ accepts. Then, using A, we can decide whether $x \in L$ in space $O(n^{1/250})$, which contradicts the fact that $L \notin SPACE(n)$. ∎