Announcements

Reminders:
- Midterm 1 during February 20 writing session.
- Solution session on Sunday, regrades due Wednesday

Review

(Un)decidability and reductions:
- A **decider** is a TM that halts on all inputs.
- A language $L$ is **undecidable** if there is no TM $M$ that halts on all inputs such that $M(x)$ accepts if and only if $x \in L$.
- A language $A$ **reduces** to $B$ if it is possible to decide $A$ given oracle access to a subroutine that decides $B$, Denote this as $A \leq_T B$ or simply $A \leq B$ (read: $B$ is at least as hard as $A$)

Time Complexity and big-Oh:
- The running time of an algorithm $A$ is a function $T_A : \mathbb{N} \to \mathbb{N}$ defined by $T_A(n) = \max_{I \in S} \{\text{number of steps } A \text{ takes on } I\}$, where $S$ is the set of instances $I$ of size $n$.
- For $f, g : \mathbb{N}^+ \to \mathbb{R}^+$, we say $f(n) = O(g(n))$ if there exist constants $c, n_0 > 0$ such that $\forall n \geq n_0$, we have $f(n) \leq cg(n)$.
- For $f, g : \mathbb{N}^+ \to \mathbb{R}^+$, we say $f(n) = \Omega(g(n))$ if there exist constants $c, n_0 > 0$ such that $\forall n \geq n_0$, we have $f(n) \geq cg(n)$.
- For both of the above, your choice of $c$ and $n_0$ cannot depend on $n$.
- For $f, g : \mathbb{N}^+ \to \mathbb{R}^+$, we say $f(n) = \Theta(g(n))$ if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.

Can’t tell if one is regular

Prove that the following language is undecidable where $M$ refers to a TM:

$$\text{REGULAR} = \{\langle M \rangle : L(M) \text{ is regular}\}.

We show that $\text{REGULAR}$ is undecidable via a reduction from $\text{HALTS}$. Suppose $M_{\text{REG}}$ decides $\text{REGULAR}$. We define a decider for $\text{HALTS}$ as follows:

```python
def M_HALTS(<M, x>):
  """def HELP(w):
    if w in {0^n1^n | n a natural number}:
      ACCEPT
    else:
```
M(x)
ACCEPT"
return M_REG(<HELP>)

Proof of correctness:
Suppose that \( M(x) \) halts, then \( L(HELP) = \Sigma^* \), so \( M\_REG(\langle HELP \rangle) \) accepts, as desired.
Suppose that \( M(x) \) loops, then \( L(HELP) = \{0^n1^n | n \in \mathbb{N} \} \), so \( M\_REG(\langle HELP \rangle) \) rejects, as desired.
Thus, we’ve shown that \( HALTS \leq REGULAR \), so \( REGULAR \) is undecidable.

O, I Think I Understand Asymptotics Now
Let \( f, g, h \) be functions from \( \mathbb{N} \) to \( \mathbb{N} \). Prove or disprove the following:
(a) If \( f = O(g) \) and \( g = O(h) \), then \( f = O(h) \)

We know by definition that there exist \( n_1 \) and \( c_1 \) such that for all \( n \geq n_1 \), \( f(n) \leq c_1 g(n) \).
Similarly, we have \( n_2 \) and \( c_2 \) for \( g \) and \( h \).
Choose \( n_0 = \max(n_1, n_2) \) and \( c = c_1 c_2 \) and let \( n \geq n_0 \).
Then, \( f(n) \leq c_1 g(n) \leq c_1 c_2 h(n) = ch(n) \), as desired, so \( f = O(h) \)

(b) If \( f = O(g) \), then \( g = O(f) \)

Let \( f(n) = 1 \) and \( g(n) = n \).
While \( f = O(g) \), there is no \( n_0, c \) you can pick that makes \( n \leq c \) for all \( n \geq n_0 \), as we just let \( n > \max(c, n_0) \) as a counterexample.

(c) \( f = O(g) \) or \( f = \Omega(g) \)

Let \( f(n) = 1 \) if \( n \) is even and \( n \) if \( n \) is odd.
Let \( g(n) = n \) if \( n \) is even and \( 1 \) if \( n \) is odd.
By the same argument as the previous problem, if we attempt to choose \( c, n_0 \) for either \( O \) or \( \Omega \), we will be able to find a sufficiently large \( n \) (even or odd as necessary) to disprove the inequality.

Bits and Pieces
Determine which of the following problems can be computed in worst-case polynomial-time, i.e. \( O(n^k) \) time for some constant \( k \), where \( n \) denotes the number of bits in the binary representation of the input.
If you think the problem can be solved in polynomial time, give an algorithm in pseudo-code, explain briefly why it gives the correct answer, and argue carefully why the running time is polynomial. If you think the problem cannot be solved in polynomial time, then provide a proof.

(a) Give an input positive integer \( N \), output \( N! \).

To represent the value of \( N! \), we will need \( \log_2(N!) \) bits. As shown in the course notes, \( \log_2(N!) = \Theta(N \log N) \). Just writing this information down will take exponential time in input-size which is \( O(\log N) \).

(b) Given as input a positive integer \( N \), output True if \( N = M! \) for some positive integer \( M \).
The polynomial-time algorithm is as follows: starting \( M \) as 1, we keep creating \( M! \) by multiplying \( M \) each time in the loop. Then, in every loop, we check whether \( M! < N \). When \( M! \geq N \), we exit the loop and check whether \( M! = N \) or \( M! > N \). If they are equal, then return true, and false otherwise.

The analysis of the algorithmic complexity is as follows: First, the number of loops will be bounded by \( O(\log N) \) because after \( M > 2 \), in each loop, we will multiply the number by two every time, so the number of loops will be terminated after \( \log_2 N \) steps. Then, in every loop, we will multiply the numbers \( x \) and \( M \), which is bounded by \( O(\log^2 N) \) by naive multiplication. Thus, the whole while loop will take \( O(\log^3 N) \). Lastly, the comparison between two numbers \( x \) and \( N \) will take \( O(\log N) \) time.

Thus, the whole algorithm takes \( O(\log^3 N) = O(n^3) \) time, which is polynomial.

(c) Given as input a positive integer \( N \), output True iff \( N = M^2 \) for some positive integer \( M \).

We will perform binary search for the value of \( x \) such that \( x^2 = N \). At the beginning of each iteration of the while loop, the invariant \( l \leq \sqrt{N} \leq h \) will be maintained. Since \( l \) and \( h \) are integers, and \( h - l \) is decreasing (see below), we will find \( \sqrt{N} \) if it is also an integer.

To analyze the run-time, the while loop will run for \( O(\log N) \) steps. Observe that \( h - l \) decreases by at least a half at the end of each iteration of the while loop. Since both \( l \) and \( h \) are integers, this process can go on at most \( \log_2 (h - l) + 1 = O(\log(N)) \) steps. During each iteration (lines 4-10), we compute the product of two integers each of which is at most \( N \). Hence, we can do this computation in \( O(\log^2 N) \) time. We also perform some comparisons and assignments, each of which only takes \( O(\log N) \) time. Thus the total time, is \( O(\log^3 N) \) time which is polynomial in \( n = \log N \).

(Extra) Lose All Scripted Responses. Improvisation Only

Let \( \text{FINITE} = \{\langle M \rangle : M \text{ is a TM and } L(M) \text{ is finite}\} \) and \( \text{TOTAL} = \{\langle M \rangle : M \text{ halts on all inputs}\} \).

Show that \( \text{TOTAL} \leq_T \text{FINITE} \).

Suppose that \( M_{\text{FINITE}} \) decides \( \text{FINITE} \).

We define a decider for \( \text{TOTAL} \) as follows, where \(<\) means lexicographically smaller.

```python
def M_TOTAL(<M>):
    <HELP> =
    """def HELP(w):
        for y <= w:
            M(y)
            ACCEPT"
    return not M_FINITE(<HELP>)
```

Proof of correctness:

Suppose \( M \) is total and halts on all inputs, then \( L(HELP) = \Sigma^* \), so \( M_{\text{FINITE}}(\langle HELP \rangle) \) will reject, as desired.

Suppose \( M \) is not total and let \( w \) be the lexicographically smallest string that \( M \) does not halt on. Then \( HELP \) will not accept any string lexicographically greater (or equal to) than \( w \), as \( M(w) \) will not halt. Thus, since there are only finitely many strings lexicographically smaller than \( w \), \( L(HELP) \) is finite. Thus, \( M_{\text{FINITE}}(\langle HELP \rangle) \) accepts, as desired.

(Extra) Asymptotically super sub

Name a function \( f(n) \) which is asymptotically super-polylogarithmic, i.e., \( f(n) = \Omega(\log^c n) \) for any constant \( c > 1 \), and at the same time asymptotically sub-polynomial, i.e., \( f(n) = O(n^\epsilon) \) for any
constant $\epsilon > 0$.

Let $f(n) = 2^{\log n}$.

- **WTS:** $\forall c > 1 : f(n) = \Omega(\log^c n)$
  Let $c > 1, d > 0$ be arbitrary numbers.
  Let $x = \log n$.

  $$\sqrt{x} = \Omega(\log x)$$

  Thus,
  $$\exists k : \forall x > k : \sqrt{x} \geq c \log x + \log(d)$$

  since $\log$ is a strictly increasing positive function.
  Substituting in $n$:
  $$\sqrt{\log n} \geq c \log \log n + \log d = \log(d \log^c n)$$

  Consequently:
  $$2^{\sqrt{\log n}} \geq d \log^c n$$

  This proves $f(n) = \Omega(\log^c n)$.

- **WTS:** $\forall \epsilon > 0 : f(n) = O(n^\epsilon)$
  Let $\epsilon > 0, d > 0$ be arbitrary numbers.
  Observe, conversely to the above case that:

  $$\sqrt{\log n} = O(\log n)$$

  Thus,
  $$\sqrt{\log n} \leq \epsilon \log(n) + \log(d) = \log(dn^\epsilon)$$

  So if we exponentiate both sides:
  $$2^{\sqrt{\log n}} \leq dn^\epsilon$$

  Thus, we've proven $\forall \epsilon > 0 : f(n) = O(n^\epsilon)$