CSC 341 - Fall 2022
Problem Set 8 Solutions

Problem 1. Prove that $\log n \in O(\sqrt{n})$.

Solution 1. It suffices to find a $c > 0$ and an $N > 0$ such that when $n \geqslant N$, we have $\log n \leqslant c\sqrt{n}$. We can pick $c = 1$ and $N = 1$.

To show why these choices work, we need to prove that $\log n \leqslant \sqrt{n}$ for $n \in \mathbb{N}$ when $n \geqslant 1$. We can do this with strong induction.

When $1 \leqslant n < 15$, we can manually check that the inequality holds. This is left as an exercise for the reader.

Now assume that $\log k \leqslant \sqrt{k}$ for $1 \leqslant k \leqslant n$. If $n < 15$, we're done. So assume $n \geqslant 15$. Thus we have

$$\begin{aligned}\log n + 1 &= 1 + \log \frac{n+1}{2} \\ &\leqslant 1 + \sqrt{\frac{n+1}{2}} \\ &= 1 + \sqrt{\frac{1}{2}}\sqrt{n+1} \\ &< 1 + \frac{3}{4}\sqrt{n+1} \\ &\leqslant \sqrt{n+1}.\end{aligned}$$

For the last step, note that when $n \geqslant 15$, $\sqrt{n+1} \geqslant 4$. Then because $\sqrt{n}$ is an increasing function, we have $1 + \frac{3}{4}\sqrt{n+1} \leqslant \frac{1}{4}\sqrt{n+1} + \frac{3}{4}\sqrt{n+1} = \sqrt{n+1}$. □

Problem 2. Prove that $1 \notin O(0)$.

Solution 2. We need to show for every $c > 0$ and $N > 0$ that $1 > c \cdot 0$. Clearly, the choice of N doesn't matter since n does not appear in the inequality. Moreover, $c \cdot 0 = 0 < 1$, so we're done. □

Problem 3. Prove the following implication using the non-limit definition of big-oh. If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$, then $f_1 f_2 \in O(g_1 g_2)$.

Solution 3. Let $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$. By definition, there is a $c_1 > 0$ and $N_1 \in \mathbb{N}$ such that when $n \geqslant N_1$, $f_1(n) \leqslant c g_1(n)$. Also by definition, there is a $c_2 > 0$ and $N_2 \in \mathbb{N}$ such that when $n \geqslant N_2$, $f_2(n) \leqslant c g_2(n)$.

Pick $N = \max(N_1, N_2)$. Note that $f_1, f_2 \geqslant 0$, so we don't need to worry about sign. Then when $n \geqslant N$, we have

$$\begin{aligned}(f_1 f_2)(n) &= f_1(n) f_2(n) \\ &\leqslant (c_1 g_1(n))(c_2 g_2(n)) \\ &= (c_1 c_2)(g_1(n) g_2(n)) \\ &= (c_1 c_2)(g_1 g_2)(n)\end{aligned}$$

So pick $c = c_1 c_2$, and we're done. $\square$

Problem 4. Prove the following implication using the non-limit definition of big-oh. If $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$, then $f_1 + f_2 \in O(max(g_1, g_2))$.

Solution 4. Let $f_1 \in O(g_1)$ and $f_2 \in O(g_2)$. By definition, there is a $c_1 > 0$ and $N_1 \in \mathbb{N}$ such that when $n \geqslant N_1$, $f_1(n) \leqslant cg_1(n)$. Also by definition, there is a $c_2 > 0$ and $N_2 \in \mathbb{N}$ such that when $n \geqslant N_2$, $f_2(n) \leqslant cg_2(n)$.

Pick $N = \max(N_1, N_2)$. Then when $n \geqslant N$, we have

$$\begin{aligned} (f_1 + f_2)(n) &= f_1(n) + f_2(n) \\ &\leqslant (c_1 g_1(n)) + (c_2 g_2(n)) \\ &\leqslant (c_1 + c_2)\max(g_1(n), g_2(n)) \\ &= (c_1 + c_2)\max(g_1, g_2)(n) \end{aligned}$$

So pick $c = c_1 + c_2$, and we're done. $\square$

Problem 5. Show that the following language is in P.

$$SORTED = \{\langle A \rangle \mid A \text{ is a sorted list of natural numbers}\}$$

You may assume that each number in A is separated by a #. The numbers are encoded into binary form.

Solution 5. All we have to do is give a polynomial time deterministic algorithm to decide $SORTED$.

First note that we can compare two natural numbers (encoded into binary) in polynomial time. All we have to do is check if one number has more significant bits (non-leading zeros). If they have the same number of significant bits, we compare most significant bits. If the most significant bits are equal, we mark them off and repeat the comparison. If we run out of bits, they're equal and we accept.

So our Turing machine S for deciding $SORTED$ is as follows.

```
S = "On input ⟨A⟩,
  1. Check if there are at least two unprocessed numbers remaining
  2. If not, accept
  3. Compare the first unprocessed number with the next
  4. If they are out of order, reject
  5. If not, mark the first unprocessed number as processed
  6. Goto (1)"
```

Clearly, S decides $SORTED$. Moreover, it performs at most a linear number of comparisons, each of which require polynomial time. Thus S runs in (deterministic) polynomial time. $\square$