Section 0 Answer Key

Richard Xu
raxu@college.harvard.edu

September 4, 2020

1. (a) Let S be the set of functions $f : [n] \to [n]$ such that $f(x) \neq x$ for any x. What is the size of S?

   (b) Let S be the set of bijective functions $f : [n] \to [n]$. What is the size of S?

   Proof. (a) For each $x \in [n]$, we have $n-1$ options for the value of $f(x)$. Therefore, $|S| = (n-1)^n$.

   (b) Notice that every injective $f : [n] \to [n]$ is also bijective. We have n options for the value of $f(0)$, $n-1$ options for $f(1)$, and so on until 1 option for $f(n-1)$. Therefore, $|S| = n!$. ☐

2. Write an algorithm for integer division. The algorithm should, on input x, y two numbers, output x/y if x is an integer multiple of y, and "error" otherwise. If x, y each have $O(n)$ digits, how many NAND operations does your algorithm take?

   Start with an inefficient algorithm. Optional challenge: write an algorithm that takes $O(n^2)$ time.

   Proof. Let A be our inefficient algorithm. A sets $s := 0, ans := 0$ and repeatedly calculates $s := s + y, ans := ans + 1$. At each step, if $s = x$ then A outputs ans, and if $s > x$ then A outputs "error".

   Correctness: we claim that $s = y \cdot ans$ at all times. This is true at the beginning, and at each step we add y to s and 1 to ans.

   Suppose $s = x$ at some point, then we know $x = y \cdot ans$ and $ans = x/y$. Then, our output is correct. Suppose s is never equal to x. Since s iterates through all integer multiples of y, x is not a multiple of y. Then, output is correct.

   Efficiency: Since it takes x/y steps before $s \geq y$, and $x/y < x$, the algorithm takes $O(x) = O(2^n)$ steps.

Optional: Efficient algorithm. Note. There are two possible algorithms: binary search, or long division. We will write the long division algorithm here. The proof of correctness is somewhat difficult. I am just trying to show that the long division we learned in school works :)

Let B be our algorithm. Suppose x has length n bits and y has length m bits. If $n \leq m$ we can check whether $x = y$. If so, output 1 and otherwise output "error".

Suppose $n < m$. Then, let z be the first m bits of x and initialize an empty string ans. If $z \geq y$, append 1 to ans and subtract y from z. If $z < y$, append 0 to ans. Then, append the next digit of x to z until we run out of digits.

At the end, if $z = 0$ then B outputs ans. Otherwise, B outputs "error".

Correctness: This is somewhat tricky. Suppose that we just finished the iteration which appended the i-th digit to x. Let w_i be the number formed by the first i digits of x, and ans_i, z_i be the values of ans, z at that point. We claim that ans_i, z_i are the quotient and remainder when we divide w_i by y.

In the first iteration, since y has m digits and z starts with m digits, $z < 2y$ and $z - y < y$. Therefore, out computation ensures that ans, z are the quotient and remainder.

Notice that $w_i = 2 \cdot w_{i-1} + x_i$. Since $w_{i-1} = ans_{i-1} \cdot y + z_{i-1}$, we have $w_i = (2 \cdot ans_{i-1}) \cdot y + (2 \cdot z_{i-1} + x)$. The second value is exactly the value of z at the start of the i-th iteration. Then, the i-th iteration ensures that ans_i, z_i are quotient and remainder when w_i is divided by y.

Since $w_n = x$, in the end ans, z are the quotient and remainder of x divided by y. If $z = 0$, then $y|x$ and we output x/y. If $z \neq 0$, then x is not a multiple of y and we output "error". Therefore, the algorithm is correct.

Efficiency: In each iteration, we compare z with y, which takes $O(n)$ operations. Since we have $O(n)$ iterations, the algorithm takes $O(n^2)$ operations. $\square$