CMPT 307 : Outline Solutions for Practice Problems (Chapter 1)

1.1.

In base b representation, the value of a single digit number is at most $b-1$, while a two digits number can express a value as large as b^2-1. It can be easily shown that $b^2-1 \geq 3(b-1)$ for $b \geq 2$.

1.3.

The depth of a d-ary tree is minimum when every node has exact d children. In that case, $n = d^0 + d^1 + d^2 + \cdots + d^D$ where D is the depth. Note that $d^0 + d^1 + d^2 + \cdots + d^D \leq d^{D+1}$ for $d > 1$. So $n \leq d^{D+1}$ yields $D \geq \log_d n - 1 = \Omega\left(\frac{\log n}{\log d}\right)$.

1.5.

Suppose $n = 2^k$.

To conclude $\sum_{i=1}^{n} \frac{1}{i} = O(\log n)$:

$$\begin{aligned}\sum_{i=1}^{n} \frac{1}{i} &= 1 + \left(\frac{1}{2} + \frac{1}{3}\right) + \left(\frac{1}{4} + \frac{1}{5} + \frac{1}{6} + \frac{1}{7}\right) + \cdots + \left(\frac{1}{2^{k-1}} + \cdots + \frac{1}{2^k - 1}\right) + \frac{1}{2^k} \\ &\leq 1 + \left(\frac{1}{2} + \frac{1}{2}\right) + \left(\frac{1}{4} + \frac{1}{4} + \frac{1}{4} + \frac{1}{4}\right) + \cdots + \left(\frac{1}{2^{k-1}} + \cdots + \frac{1}{2^{k-1}}\right) + \frac{1}{2^k} \\ &\leq 1 + k \\ &= O(\log n)\end{aligned}$$

To conclude $\sum_{i=1}^{n} \frac{1}{i} = \Omega(\log n)$:

$$\begin{aligned}\sum_{i=1}^{n} \frac{1}{i} &= 1 + \left(\frac{1}{2}\right) + \left(\frac{1}{3} + \frac{1}{4}\right) + \left(\frac{1}{5} + \frac{1}{6} + \frac{1}{7} + \frac{1}{8}\right) + \cdots + \left(\frac{1}{2^{k-1}+1} + \cdots + \frac{1}{2^k}\right) \\ &\geq 1 + \left(\frac{1}{2}\right) + \left(\frac{1}{4} + \frac{1}{4}\right) + \left(\frac{1}{8} + \frac{1}{8} + \frac{1}{8} + \frac{1}{8}\right) + \cdots + \left(\frac{1}{2^k} + \cdots + \frac{1}{2^k}\right) \\ &\geq 1 + k\left(\frac{1}{2}\right) \\ &= \Omega(\log n)\end{aligned}$$

1.7.

Consider multiplying a n-bit number x with a m-bit number y using the recursive algorithm. Observe the followings:

(a) The algorithm terminates after m recursive calls.
(b) Each recursive call takes $O(n)$ time.

To see (a), think of division by 2 as bit shifting. Therefore, the total running time is $O(m \cdot n)$.

1.10.

$a \equiv b \bmod N$ means N divides $a - b$ (you may think of it as $a - b \equiv 0 \bmod N$). Since M divides N, M also divides $a - b$, which yields $a \equiv b \bmod N$.

1.12.

Note that $a^b \equiv (a \bmod N)^b \bmod N$. Then plug $2^{2^{2006}} = 2^{2 \cdot 2^{2005}} = 4^{2^{2005}}$ into the formula yields the answer as $2^{2^{2006}} \equiv 1 \bmod 3$.

1.15.

Consider the condition $\gcd(x, c) = 1$.

For sufficiency, observe that under the condition $\gcd(x, c) = 1$ (which implies $x \not\equiv 0 \bmod c$), if $ax \equiv bx \bmod c$ (which implies $(a - b)x \equiv 0 \; mod \; c$), then $a - b \equiv 0 \bmod c$.

For necessity, suppose that $\gcd(x, c) > 1$, say $x = mq$ and $c = nq$ for $m, n \geq 1, q > 1$. Then we can find some a, b (e.g. a, b that satisfy $a - b \equiv n \bmod c$) such that $ax \equiv bx \bmod c$, but $a \not\equiv b \bmod c$.

1.17.

For the iterative algorithm, it performs $y - 1$ iterations. In the i^{th} iteration, it multiplies x^i with x, which takes $O\big((\log(x^i)) \cdot \log(x)\big) = O(i \log^2 x)$. So the total running time is $O(\log^2 x + 2 \log^2 x + \cdots + (y - 1) \log^2 x) = O(y^2 \log^2 x)$.

For the recursive algorithm, it performs $O(\log y)$ iterations. In the i^{th} iteration, it multiplies $x^{y/2^i}$ with itself, which takes $O\left(\log^2\left(x^{y/2^i}\right)\right) = O\left(\frac{1}{2^{2i}} \cdot y^2 \log^2 x\right)$. So the total running time is $\left(\left(\sum_{i=1}^{\log y} \frac{1}{2^{2i}}\right) \cdot y^2 \log^2 x\right) = O(y^2 \log^2 x)$ since $\sum_{i=1}^{\infty} \frac{1}{2^{2i}} = O(1)$.

The two algorithm have running times of the same order.

1.19.

This can be shown by Induction. The condition is clearly holds for $n = 1$. Now assume that it holds for $n \leq k$. We need to show that it also holds for $n = k + 1$. Note that

$$\gcd(F_{k+1}, F_{k+2}) = \gcd(F_{k+1}, F_{k+2} - F_{k+1}) = \gcd(F_{k+1}, F_k) = 1$$

where the first equality is by Euclid's rule, the second equality is by the definition of Fibonacci number, and the last equality is by the induction hypothesis.

1.25.

First use Fermat's Little Theorem to deduce that 2^{125} is the inverse of 2 mod 127. Then by observation, 64 is the inverse of 2 mod 127, so $2^{125} \equiv 64 \bmod 127$ by the uniqueness of inverse.

1.39.

Suppose that $b^c = q(p-1) + r$. Then

$$a^{b^c} \equiv a^{q(p-1)+r} \equiv a^r \bmod p$$

where the second congruence is by Fermat's Little Theorem.

To compute $a^r \bmod p$, We first compute the exponent $r \equiv b^c \, mod \, (p-1)$, and then we compute $a^r \bmod p$. Both steps can be computed efficiently via modular exponentiations.