CSE 160A, Discrete Mathematics, Winter 1993
Homework 3, Answer Key

Short problems:

If $c_1 = 2$ and $c_{k+1} = 2c_k$, give an explicit formula for c_k. Answer: $c_i = 2^i$, since $c_1 = 2^1$ and if $c_i = 2^i$, then $c_{i+1} = 2 * c_i = 2 * 2^i = 2^{i+1}$.

If $c_1 = 2$ and $c_{k+1} = (c_k)^2$, give an explicit formula for c_k. Write $c_i = 2^b_i$. Then $c_{i+1} = {c_i}^2 = (2^b_i)^2 = 2^{2b_i} = 2^{b_{i+1}}$. Thus, the b_i's are doubling every time, so they will be on the order of 2^i, like in the last problem. Since $b_1 = 1 = 2^0$, $b_i = 2^{i-1}$. So we guess $c_i = 2^{2^{i-1}}$. This guess is easily verified by induction: $c_1 = 2 = 2^{2^0}$, if $c_i = 2^{2^{i-1}}$, then $c_{i+1} = {c_i}^2 = 2^{2^{i-1}2} = 2^{22^{i-1}} = 2^{2^i}$.

Full Proofs: (4 pts. ea.)

- Section 4.2, pp. 204-205: 2

  Show that for any $k \geq 8$ k cents can be obtained with a sum of 5 and 3 cent stamps.

  Base Case: (k=8; Goal: 8 cents can be obtained by 5 and 3 cent stamps.) 8 cents can be made with 1 5-cent and 1 3-cent stamp.

  Induction Step: (Goal: Assuming true for some $k \geq 8$, prove it is still true for $k+1$.)

  Let k cents be made with x 5 cent stamps and y 3-cent stamps, i.e. $k = 5x + 3y$ for non-negative integers x, y. Then if $x \neq 0$, trade in one of the 5 cent stamps for 2 3 cent stamps, i.e. let $x' = x - 1, y' = y + 2$. Then $5x' + 3y' = 5(x-1) + 3(y+2) = 5x + 3y + 1 = k + 1$. If $x = 0$, then, since $k \geq 8$, $y \geq k/3 > 2$, so there are at least 3 3-cent stamps. Trade in 3 3-cent stamps for 2 five cent stamps, i.e. let $x' = x + 2, y' = y - 3$. Then $5x' + 3y' = 5(x+2) + 3(y-3) = 5x + 3y + 1 = k + 1$. In either case, we have expressed $k+1$ cents as x' 5-cent and y' 3-cent stamps.

  Thus, by induction, any amount of postage greater than 8 cents can be made with 5 and 3 cent stamps.

- Section 4.2, pp. 204-205: 11

  Prove that $\forall$ (positive integers) n, $\sum_{i=1}^{i=n} \frac{1}{i(i+1)} = \frac{n}{n+1}$.

  First, for $n = 1$, we have $sum_{i=1}^{i=1} \frac{1}{i(i+1)} = \frac{1}{1(2)} = 1/2 = \frac{1}{1+1}$, so the claim is true for $n = 1$.

  Assume the equation is true for $n = k$, for some positive integer k, in other words, that $\sum_{i=1}^{i=k} \frac{1}{i(i+1)} = \frac{k}{k+1}$. Then $\sum_{i=1}^{i=k+1} \frac{1}{i(i+1)} = \sum_{i=1}^{i=k} \frac{1}{i(i+1)} + 1/(k+1)(k+2) = \frac{k}{k+1} + \frac{1}{(k+1)(k+2)} = \frac{(k)(k+2)+1}{(k+1)(k+2)} = \frac{k^2+2k+1}{(k+1)(k+2)} = \frac{(k+1)^2}{(k+1)(k+2)} = \frac{k+1}{k+2}$. So if the equation is true for $n = k$, then it is also true for $n = k+1$.
  Thus, by induction, since the equation holds for $n = 1$, and if it holds for $n = k$ then it also holds for $n = k+1$, the equation is true for all integers $n \geq 1$.

- Section 4.3, pp. 210-212: 12

  Prove, for any $n \geq 1$, $5 | (7^n - 2^n)$.

  Proof: First, for $n = 1$, we have $5 | 5 = 7^1 - 2^1$, so the claim is true for $n = 1$.

  Assume the claim is true for $n = k$, i.e. $5 | (7^k - 2^k)$. Then $7^k - 2^k = 5z$ for some integer z, or $7^k = 5z + 2^k$. Then $7^{k+1} = 7 * 7^k = 35z + 72^k = 35z + 52^k + 22^k = 5(7z + 2^k) + 2^{k+1}$. so $7^{k+1} - 2^{k+1} = 5(7z + 2^k)$, and so is divisible by 5.

  Thus, by induction, $5 | (7^n - 2^n)$ for all $n \geq 1$.

- Section 4.3, pp. 210-212: 17

  Prove, for all integers $n \geq 0$, that $2^n < (n+2)!$.

  First, for $n = 0$, we have $2^0 = 1 < 2 = (0+2)!$, so the claim is true for $n = 0$.

  Assume the claim is true for $n = k \geq 0$, i.e. that $2^k < (k+2)!$. Then $2^{k+1} = 2 * 2^k < (k+3)2^k < (k+3)(k+2)! = (k+3)!$.

  Thus, if the claim is true for $n = k$ then it is also true for $n = k+1$, so by induction it is true for all n.

- Section 4.4., pp. 219-221: 2 Define a sequence by: $b_1 = 3, b_2 = 6, b_k = b_{k-1} + b_{k-2}$ for $k \geq 3$. Prove $3|b_n$ for all values of $n \geq 1$.

  First, for $n = 1$, $3|3 = b_1$. For $n = 2$, $3|6 = b_2$.

  Second, assume that the claim were true for all values of n between 1 and k, where $k \geq 2$; i.e. assume, for all $1 \leq n \leq k$, $3|b_n$. Then we'll show it also holds for $n = k+1$, i.e that $3|b_{k+1}$.

  Applying the assumption to $n = k-1$ and $n = k$ (note that since $k \geq 2$, $k-1 \geq 1$), we know $3|b_{k-1}$ and $3|b_k$. Therefore $b_{k-1} = 3x$ for some integer x and $b_k = 3y$ for some integer y. Then $b_{k+1} = b_k + b_{k-1} = 3x + 3y = 3(x+y)$, so $3|b_{k+1}$.

  Thus our claim is true for $n = 1$ and $n = 2$, and for any $k \geq 2$, if it is true for $1 \leq n \leq k$, then it is also true for $n = k+1$. Thus, by strong induction, it is true for all values of n.

- Section 4.4., pp. 219-221: 10 The gap is that the induction step implicitly assumes $k \geq 2$. If $k = 1$, the induction step re-writes $r^k = r^1$ as $r^1 = (r^0)(r^0)/r^{-1}$. Since the base case was $k = 0$, we know that $r^0 = 1$, but not that $r^{-1} = 1$, since we are only proving this for $k \geq 0$. Thus, if we wanted to complete the proof, we would need a separate base case for $k = 1$. Of course, this base case would attempt to prove something false, that $r^1 = r = 1$ no matter what r is. The argument in the induction step in the previous problem also assumed $k \geq 2$. That is why we included both $k = 0$ and $k = 1$ as base cases.

- Prove that for F_k the k'th Fibonnaci number, $F_k \leq 2^k$ for all $k \geq 0$. (Hint: use strong induction; see Epp 8.1. for the def. of Fibonacci number.)

  First for $k = 0$, we have $F_0 = 1 \leq 1 = 2^0$, so the claim is true for $k = 0$. For $k = 1$, we have $F_1 = 1 \leq 2 = 2^1$, so the claim is true for $k = 1$.

  Let $n \geq 1$ be an integer. Assume that $F_k \leq 2^k$ for all $0 \leq k \leq n$, In particular, $F_n \leq 2^n$ and $F_{n-1} \leq 2^{n-1}$. Then $F_{n+1} = F_n + F_{n-1} \leq 2^n + 2^{n-1} = 2^n + 1/22^n = (1.5)2^n < 22^n = 2^{n+1}$. Thus, if the inequality holds for all $1 \leq k \leq n$, it also holds for $n+1$.

  Thus, by strong induction, it holds for all n.

- A database is keeping track of k "accounts", labelled 1 to k, which at any time have some integer value; write $A_i(t)$ to mean the value of account i after t transactions have been made. The t'th "transaction" is of the form: add $v(t)$ to account i_t. Thus, $A_i(t+1) = A_i(t) + v(t)$ if $i = i_{t+1}$, $A_i(t+1) = A_i(t)$ otherwise. For reasons of fault-tolerance, in addition to a memory location A_i being kept for each account, an additional register Sum is kept (to allow recovery if one of the memory locations is erased or damaged). For each transaction t, the database adds $v(t)$ to both the register A_{i_t} and to Sum; we call the value of Sum after the t'th transaction $Sum(t)$. Initially, all accounts and Sum are set to 0. Prove that, for any t, after t transactions, we have $Sum(t) = \sum_{i=1}^{i=k} A_i(t)$.

  We prove this by induction on the number t of transactions.

  At $t = 0$, $Sum(0) = 0 = \sum_{i=1}^{i=k} 0 = \sum_{i=1}^{i=k} A_i(0)$, so the equation is true at $t = 0$.

  Assume the equation holds after t_0 transactions have been made, i.e., $Sum(t_0) = \sum_{i=1}^{i=k} A_i(t_0)$. Then after the next transaction, t_0+1, we have $Sum(t_0+1) = Sum(t_0) + v(t_0+1) = \sum_{i=1}^{i=k} A_i(t_0) + v(t_0+1) = \sum_{i=1}^{i=i_{t_0+1}-1} A_i(t_0) + \sum_{i=i_{t_0+1}+1}^{i=k} A_i(t_0) + A_{i_{t_0+1}} + v(t_0+1)$ (here, we just break up a sum into three parts, the part before i_{t_0+1}, the part after, and $i = i_{t_0+1}$) $= \sum_{i=1}^{i=i_{t_0+1}-1} A_i(t_0+1) + \sum_{i=i_{t_0+1}+1}^{i=k} A_i(t_0+1) + A_{i_{t_0+1}}(t_0+1)$ (here, we use the recursive definition of $A_i(t+1)$ given in the problem.) $= \sum_{i=1}^{i=k} A_i(t_0+1)$ (rearranging the sum again). Thus, if the equation holds after t_0 transactions, it will still hold after t_0+1 transactions.

  Thus, by induction, the equation holds after any number of transactions t.

- Consider the algorithm described by the following recursive "pseudo-code" program:

  1. Program: RPM {Russian Peasant Multiplication}
  2. Inputs: X,Y: Integer

3. If $X = 0$ Return 0;
4. If $X > 0$ and $X mod 2 = 0$ Return $2 * RPM(X/2, Y)$
5. If $X > 0$ and $X mod 2 = 1$ Return $RPM(X-1, Y) + Y$

Prove, by strong induction on X, that on any input X, Y with $X \geq 0$, $RPM(X, Y) = X * Y$.

The proof is by strong induction on X. Let Y be any integer.

First, for $X = 0$, by the program definition, $RPM(0, Y) = 0 = 0 * Y$. (line 3) So the claim is true for $X = 0$.

Let $k \geq 1$ be an integer. Assume that , for all values of X with $0 \leq X \leq k-1$, the claim is true, i.e. $RPM(X, Y) = XY$.

Case 1: If k is even, then $RPM(k, Y) = 2RPM(k/2, Y)$. Appplying the induction assumption to $X = k/2$, we have $RPM(k/2, Y) = k/2 * Y$, so $RPM(k, Y) = 2(k/2Y) = kY$.

Case 2: If k is odd, then $RPM(k, Y) = RPM(k-1, Y) + Y$. Appplying the induction assumption to $X = k-1$, we have $RPM(k-1, Y) = (k-1) * Y$, so $RPM(k, Y) = (k-1)Y + Y = kY$.

In either case, $RPM(k, Y) = kY$.

Thus, by strong induction, $RPM(X, Y) = X * Y$ for all integers $X \geq 0$.

- Consider the algorithm described by the following "pseudo-code" program:

```
1. Program:1B
2. Input:x
3. begin;
4. y:= 0;
5. While x > 0 do:
6. beginwhile;
7.    If x MOD 2 =1 THEN x:= x-1; y := y+1;
8.                 ELSE x:= x/2;
9. endwhile;
10. Output y;
11. end;
12.
```

Prove that for any positive integer n, when given input $x = n > 0$, the while loop is executed at most $2(\log_2 n) + 1$ times.

Hint: One approach: use a proof by strong induction on the value of x. Second approach:use a normal induction proof on t to show that if the program has not halted before $2t$ loops, then after the $2t$'th loop, the value of x is at most $n/2^t$. Then use this to prove the time bound.

We'll show the proof by strong induction. Let $T(x)$ represent the number of loops the program makes on input x. We wish to prove $T(X) \leq 2log_2 X + 1$ for all integers $X \geq 1$.

First, if $X = 1$, we see that on input 1, the program loops once, subtracting 1 from X to get 0, at which point it stops. Therefore, $T(1) = 1 \leq 2 * 0 + 1 = 2(log_2 1) + 1$.

Let $k \geq 2$ be an integer. Assume, for any integer $1 \leq X \leq k-1$, that $T(X) \leq 2 \log_2 X + 1$.

Case 1: If k is even, the program divides $X = k$ by 2 in the first loop. Thus, the number of loops is one more than the number of loops for $X = k/2$, i.e. $T(k) = 1 + T(k/2)$. Applying the induction assumption to $X = k/2$, we have: $T(k) = 1 + T(k/2) \leq 1 + 2\log_2(k/2) + 1 = 2(\log_2 k - 1) + 2 = 2\log_2(k) \leq 2\log_2(k) + 1$.

Case 1: If k is odd, the program subtracts 1 from $X = k$ in the first loop. Then, the value of X is $k-1$ which is even. So in the second loops this value is halved to $(k-1)/2$. Thus, the number of loops on $X = k$ is two more than the number of loops for $X = (k-1)/2$, i.e. $T(k) = 1 + T((k-1)/2)$. Applying the induction assumption to $X = (k-1)/2$, we have: $T(k) = 2 + T((k-1)/2) \leq 2 + 2\log_2((k-1)/2) + 1 = 2(\log_2(k-1) - 1) + 3 \leq 2\log_2(k-1) + 1 \leq 2\log_2 k + 1$.

Thus, in either case, $T(k) \leq 2log_2(k) + 1$..

Thus, by strong induction, $T(X) \leq 2log_2(X) + 1$ for all $X \geq 1$.