Base conversion: 20 points Give an efficient algorithm to convert integers from base 3 to binary. Your algorithm should be better than the $O(n^2)$ method ala the calibration homework. You can use the $O(n \log 3)$ Kurasawa binary multiplication algorithm from class as a subroutine.

Multiplication in Three Pieces: 20 pts. In class, we saw a divide and conquer algorithm for multiplication that divided each $n$ bit integer into high and low positions, each $n/2$ bits long. Consider algorithms that break the integers up into three pieces instead, the high order, mid order, and low order pieces, each $n/3$ bits long. What is the best divide-and-conquer multiplication algorithm of this type you can find? Is it better or worse than the two piece algorithm from class?

Triangles: 20 pts On the calibration homework, we saw an $O(nm)$ algorithm to compute whether a graph $G$ had a triangle, three distinct nodes $x, y, z$ so that any two were connected by an edge in $G$. For large $m$, this is $O(n^3)$. Use the Strassen Matrix Multiply algorithm in the text as a subroutine to give a faster algorithm for this problem, assuming the graph $G$ is presented in adjacency matrix form.

Binary Tree Isomorphism: 20 points Consider the following recursive algorithm, which makes the following assumptions. $x, y$ are the roots of two binary trees, $T_x$ and $T_y$. Left$(z)$ is a pointer to the left child of node $z$ in either tree, and Right$(z)$ points to the right child. If the node doesn’t have a left or right child, the pointer returns “NIL”. Each node $z$ also has a field Size$(z)$ which returns the number of nodes in the sub-tree rooted at $z$. Size$(NIL)$ is defined to be 0. The algorithm SameTree$(x, y)$ returns a boolean answer that says whether or not the trees rooted at $x$ and $y$ are the same if you ignore the difference between left and right pointers.

1. Program: SameTree(x,y: Nodes): Boolean;
2. IF Size$(x) \neq$ Size$(y)$ THEN return False; halt.
3. IF $x = NIL$ THEN return True; halt.
4. IF (SameTree(Left$(x)$, Left$(y)$) AND SameTree(Right$(x)$, Right$(y)$))
   OR (SameTree(Right$(x)$, Left$(y)$) AND SameTree(Left$(x)$, Right$(y)$))
   THEN return True; halt.
5. Return False; halt.

Give a time analysis (up to order) for this program for the case when the trees rooted at $x$ and $y$ are both complete balanced trees with $n$
nodes. (Every node $z$ in a complete balanced tree has $\text{Size}(\text{Left}(z)) = \text{Size}(\text{Right}(z))$.) Then prove that the analysis still holds when the trees are not perfectly balanced.

**Implementation: 20 pts** Implement the grade-school and clever divide-and-conquer multiplication algorithms, where the input is coded as an array of digits. Plot both performances on a log-log scale, for random integers of length $n$ for $n$ different powers of 2. Then combine them to use a threshold $T$ as follows: IF $n < T$ use Gradeschool Multiply ELSE use the Divide-and-Conquer recurrence (but recursive calls are to the thresholded algorithm). What is the value of $T$ that gives the best performance on random $n$ digit numbers? Is it bigger or smaller than the cross-over point where divide-and-conquer beats gradeschool multiplication? Show the data to support your conclusions.