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?

Least Common Ancestor: 20 points Consider the following recursive algorithm that takes as input a binary tree $T$.

Each non-leaf in $T$, $x$, has left-child $x.left$, and right child $x.right$, and each non-root has parent $x.parent$. (Child pointers at leaves and the parent pointer at the root return NIL). It uses a depth-first search procedure $DFS$ that is linear-time in the size of the sub-tree and returns the list of nodes in the sub-tree. It computes, for each pair of nodes $x$ and $y$ in $T$, the deepest node that is an ancestor of both $x$ and $y$, and stores it in an array $LCA[x, y]$. The main idea is that if $x$ is in the left sub-tree of the root, and $y$ is in the right sub-tree, then the least common ancestor is the root. Otherwise, the least common ancestor is in the sub-tree that contains both $x$ and $y$.

LeastCommonAncestor$(r: \text{node})$

1. $LCA[r, r] \leftarrow r$
2. IF $r.left \neq \text{NIL}$ THEN
3. LeastCommonAncestor$(r.left)$;
4. $L_1 \leftarrow DFS(r.left)$;
5. IF $r.right \neq \text{NIL}$ THEN
6. LeastCommonAncestor$(r.right)$;
7. $L_2 \leftarrow DFS(r.right)$.
8. FOR each $x \in L_1$
9. FOR each $y \in L_2$
10. $LCA[x, y] \leftarrow r$

First, give a recurrence relation for the time of this algorithm when the input is a complete binary tree of size $n = 2^d - 1$, where $d$ is the depth of the tree. (Note that such a complete binary tree is always perfectly
balanced, with left and right sub-trees of the same size.), and solve it to
give a time analysis for the algorithm in the complete binary tree case.
Then give a worst-case analysis for the time, not making any assumptions
about the input tree.

**Triangle** A triangle in an undirected graph \( G \) is a triple of nodes \( u, v, w \) so that
any two of them are adjacent in the graph. Use Strassen’s \( O(n^\log_2 7) \) time
matrix multiply algorithm to determine whether an undirected graph, in
adjacency matrix format, has a triangle, in the same order of time.

**Back-tracking: Hamiltonian path** Consider the following algorithm for de-
ciding whether a graph has a Hamiltonian Path from \( x \) to \( y \), i.e., a simple
path in the graph from \( x \) to \( y \) going through all the nodes in \( G \) exactly
once. \( (N(x) \) is the set of neighbors of \( x \), i.e. nodes directly connected to
\( x \) in \( G \).

1. \( \text{HamPath}(G, x : \text{node}, y : \text{node}) \)
2. If \( x = y \) is the only node in \( G \) return \( \text{True} \).
3. If no node in \( G \) is connected to \( x \), return \( \text{false} \).
4. For each \( z \in N(x) \) do:
5. If \( \text{HamPath}(G - \{x\}, z, y) \), return \( \text{true} \).
6. Return \( \text{false} \)

a. Explain (informally) why this algorithm is correct. (5 points) b. If
every node of the graph \( G \) has degree (number of neighbors) at most 3,
how long will this algorithm take at most? (15 points) (Hint: you can get
a tighter bound than the most obvious one.)

**Implementation: 20 pts** Implement the \( O(n^\log_2 3) \) time divide-and-conquer
multiplication algorithm from class, and the grade-school multiplication
algorithm. Plot the average times to multiply \( n \) bit numbers using the two
methods (on log-log scales). When is the divide-and-conquer algorithm
better? Then consider a hybrid algorithm, where you replace the base-
case of the recursion with the grade-school method for inputs of size less
than some threshold \( T \). Experimentally determine the best value of \( T \).