For each of the algorithm problems, design as asymptotically efficient an algorithm as possible. Give a correctness argument (explanation, if it is relatively simple, or proof if not) and time analysis. You may use any well-known algorithm or data structure, or algorithm from the text or from class, as a subroutine without needing to provide details.

**Base Conversion**. Give an algorithm that inputs an array of \( n \) base base 10 digits representing a positive integer and outputs an array of bits representing the same integer in base 2. Your algorithm should be \( o(n^2) \), strictly better than the time asked for on the calibration homework. You will probably need to use a divide-and-conquer strategy, and use a fast integer multiplication sub-routine (from class).

**Binary Tree Isomorphism** Two rooted trees \( T_1 \) and \( T_2 \) are isomorphic if there is a 1-1 onto map \( f : T_1 \rightarrow T_2 \) so that \( f(\text{root}_1) = \text{root}_2 \) and \( p_2(f(x)) = f(p_1(x)) \), for every \( x \in T_1 \) except \( \text{root}_1 \). (Here, \( \text{root}_1 \) is the root of \( T_1 \), \( \text{root}_2 \) is the root of \( T_2 \), and \( p_1, p_2 \), represent the parents in the respective trees.) On the calibration homework, an \( O(n^2) \) time algorithm for this problem was given. Give a more efficient \( (o(n^2)) \) algorithm for the same problem. (Note that the \( f \) is what your algorithm is looking for, not an input.)

**Closest pair of points in 3d** Consider the problem of finding the closest pair of distinct points among a list of points in 3-d, \( P_1,..,P_n \), where \( P_i = (x_i,y_i,z_i) \). Modify the two dimensional algorithm given in class. You may need to use a data structure such as a binary search tree or hash table.

**Dice pools**: This problem arises from calculating success probabilities for certain role-playing games, where players roll dice in proportion to their character’s abilities, and each die is either a “Success”, a “Failure” or “Neutral”, and the outcome is determined by the number of successes minus the number of failures. (For example, in one game, dice take random values from 1 to 10, with 1 being a “Failure” and 8-10 being a “Success”.) Abstractly, the problem is: there are \( n \) independent random variables, \( X_1...X_n \). Each variable is +1 with probability \( p \), -1 with probability \( q \) and 0 otherwise, where \( 0 \leq p,q \leq 1 \) and \( p + q \leq 1 \). (In the above example, \( p = 3/10, q = 1/10 \).) We want to calculate, given \( n,p,q \), an array of probabilities: for all \( k \) with \( -n \leq k \leq n \), compute the probability that \( \sum_{i=1}^{n} X_i = k \). Your algorithm should be polynomial-time in \( n \). Assume
arithmetic operations are constant time. (6 pts. correct, poly-time alg., 4 pts efficiency; my best time is $O(n \log n)$.)

**Implementation: Integer Multiplication** Implement the $O(n^{\log 3})$ divide-and-conquer algorithm for integer multiplication from class, but with a threshold, below which naive “gradeschool” multiplication is used. Use an array of digits to represent inputs and outputs. Experimentally determine the optimal threshold. For what values of $n$ do you see an improvement in the time using divide-and-conquer, both using no threshold and using the optimal threshold?