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 class, as a sub-routine without needing to provide details.

**Base Conversion**. Give an algorithm that inputs an array of \( n \) base 10 digits representing a positive integer in base 10 and outputs an array of bits representing the same integer in binary. Get as close as possible to linear time.

**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, \( p = \frac{3}{10}, q = \frac{1}{10} \).) We want to compute, given \( n \), 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 \( O(n \log n) \).)

**Weighted Median, Problem 9-2, part c., p. 194**

**Merging lists**: Consider the problem of merging \( k \) sorted lists with a total of \( n \) elements into a single sorted list. Show that in the comparison model, the complexity of this problem is \( \Theta(n \log k) \). (Note that \( k \) can be any value or function of \( n \); in particular, do not assume that it is constant. You can use the \( O(n \log n) \) lower bound for sorting without proof.)

**Implementation: Integer Multiplication** Implement the \( O(n \log^3 n) \) divide-and-conquer algorithm for integer multiplication, but with a threshold, below which naive “gradeschool” multiplication is used. 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?