Question 1. Suppose you are choosing between the following three algorithms, all of which have $O(1)$ base cases for size 1:

1. Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining the solutions in linear time.

2. Algorithm B solves problems of size $n$ by recursively solving one subproblem of size $n/2$, one subproblem of size $2n/3$, and one subproblem of size $3n/4$ and then combining the solutions in linear time.

3. Algorithm C solves problems of size $n$ by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.

What are the running times of each of these algorithms (in asymptotic notation) and which would you choose? Hint: Approach Algorithm B via the substitution method to avoid tough summations. Also, solving that one as $O(n^d)$ for the smallest integer $d$ is all I’m looking for.

Question 2. Recall the problem of finding a majority element we solved in class (Q3 of Ch. 5, p. 246) using $O(n \log n)$ yes-no equivalence tests. Now we’ll try to solve it faster. Consider the following procedure: Arbitrarily pair up the elements. Test each pair and if the two elements are different, discard both of them; if they are the same, keep just one of them. Use this procedure to build a divide-and-conquer algorithm. Analyze its running time by writing a recurrence relation and evaluating it. [Hint: Show that after one iteration of the procedure, there are at most $n/2$ elements left, and that if $A$ has a majority element, it is still the majority element in the subproblem.]

Question 3. You are responsible for compressing strings of DNA bases for the Human Genome Project, which consist of the symbols A, C, G, and T. Also, some bases have not been sequenced, and those missing bases are represented by the symbol ?. Suppose that source symbols are uncorrelated and occur with the following frequencies: A with probability 0.18, C with probability 0.12, G with probability 0.09, T with probability 0.20, and ? with the remaining probability (0.41).

(a) What encoding would Huffman codes produce for this source? What is the average number of bits per symbol that is achieved?

(b) Now suppose that there is correlation between symbols (but the frequencies remain the same). In particular, ?’s only occur in runs of length five or more and C is always followed by A. Define a new alphabet that exploits these facts, but still allows you to use Huffman codes. How many bits per symbol does your new code achieve?

(c) Why are Huffman codes no longer the best choice? Design a code that beats the best bound achievable by Huffman.

Question 4. A Toeplitz matrix is an $n \times n$ matrix $A = (a_{ij})$ such that $a_{ij} = a_{i-1,j-1}$ for $i = 2, 3, \ldots n$ and $j = 2, 3, \ldots n$.

(a) Is the sum of two Toeplitz matrices necessarily Toeplitz? What about the product of two Toeplitz matrices?

(b) Describe how to represent a Toeplitz matrix so that two $n \times n$ Toeplitz matrices can be added in $O(n)$ time.

(c) Give an $O(n \log n)$ time algorithm for multiplying an $n \times n$ Toeplitz matrix by a vector of length $n$, using your representation from part (b).

(d) Give an efficient algorithm for multiplying two $n \times n$ Toeplitz matrices. Analyze its running time.

Question 5. Chapter 6, Problem 1 on p. 312.