Question 1. Suppose you have a set of $N$ gold coins of which all but one have identical weight. The exception is a single fake coin that weighs less than real coins. To detect the fake, you have a scale which you can use to weigh any two piles of coins. The scale will either report that the two piles are of equal weight, or if not, which of the two piles is lighter.

(a) Specify an algorithm which correctly identifies the fake coin using only $\log_2 N + 1$ weighings. Argue for why your algorithm achieves this bound. Hint: Use divide-and-conquer.

(b) Now specify a better algorithm which correctly identifies the fake coin using only $\log_3 N + 1$ weighings. Argue for why your algorithm achieves this bound. (This turns out to be optimal).

Question 2. 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 three 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) (This whole question will be on Homework 4. This part is too hard for the midterm, but you can start thinking about it anyway.) Why are Huffman codes no longer the best choice? Design the best code that you can for this source to minimize average bits per symbol.

Question 3. For each of the following recurrence relations, specify a big-$O$ closed form for $T(n)$, and an algorithm that we’ve seen in class or on the homework which has a recurrence of that form. In each case, assume $T(1) = O(1)$.

(a) $T(n) = T(n/2) + O(1)$

(b) $T(n) = 2T(n/2) + O(n)$

(c) $T(n) = 2T(n/2) + O(n \log n)$

(d) $T(n) = 3T(n/2) + O(n)$


Question 5. Chapter 4, Problem 10. (Probably too hard for the midterm, but good practice for studying purposes).

Question 6. Chapter 5, Problem 3. (We might end up covering this one in class).

Question 7. I will definitely ask some questions which test your knowledge of basic algorithms we’ve learned in class and on the homework, including Stable Marriage, max sum subinterval, topological ordering, interval scheduling, Dijkstra’s, the MST algorithms, the various algorithms for multiplication, etc. With a partner, generate a representative small instance for each of these, and make sure your partner can correctly and quickly run the algorithm and generate the output.