Problem 1 (10 Points). Prove the following two properties of the Huffman encoding scheme:

(a) If some character occurs with frequency more than 2/5, then there is guaranteed to be a codeword of length 1.

(b) If all characters offuc with frequency less than 1/3, then there if guaranteed to be no codeword of length 1.

Problem 2 (5 Points). Under a Huffman encoding of n symbols with frequencies $f_1, f_2, ..., f_n$, what is the longest a codeword could possible be? Give an example set of frequencies that would produce this case.

Problem 3 (15 Points). Show that for any integer n that is a power of 2, there is an instance of the set cover problem with the following properties:

i. There are n elements in the base set B.

ii. The optimal cover uses just 2 sets.

iii. The greedy algorithm picks at least $\log_2 n$ sets.

Problem 4 (15 Points). Solve the following recurrences, giving the tightest O bound possible:

(a) $T(n) = T(n-1) + n$

(b) $T(n) = T(\sqrt{n}) + 1$

(c) $T(n) = T(n-1) + 2^n$

Problem 5 (10 Points). Solve the following recurrences using the master method, giving the tightest O bound possible:

(a) $T(n) = 2T(n/3) + 1$

(b) $T(n) = 5T(n/4) + n$

(c) $T(n) = 7T(n/7) + n$

(d) $T(n) = 9T(n/3) + n^2$

(e) $T(n) = 10T(n/3) + n^2$

Problem 6 (10 Points).

(a) Prove that it cannot be asymptotically faster to square an integer than it is to multiply two integers. That is, show that if we can square an n digit integer in time $O(n^d)$, that we can also multiply two n digit integers in time $O(n^d)$.

(b) Prove that it cannot be asymptotically faster to square a matrix than it is to multiply two matrices.

Problem 7 (5 Points). You are given a list (in either array form or linked list form) of n elements from some ordered domain, and you notice that some of the elements are duplicates; that is, they appear more than once in the list. Show how to remove all duplicates from the list in time $O(n \log n)$.

Problem 8 (10 Points). You are given two sorted arrays of with m and n elements, respectively. Give an $O(\log m + \log n)$ time algorithm for computing the kth smallest element in the union of the two lists.

Problem 9 (20 Points). A list A of n elements $a_i, ..., a_n$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $a > b$?". However, you *can* answer questions of the form: "is $a = b$?" in constant time.

(a) Show how to solve this problem in $O(n \log n)$ times. (Hint: Split the list into two lists A_1 and A_2 of half the size. Does knowing the majority elements of A_1 and A_2 help you figure out the majority element of A? If so, you can use a divide-and-conquer approach.)

(b) Can you give a linear-time algorithm? (Hint: Here's another divide-and-conquer approach:

- Pair up the elements of A arbitrarily, to get $n/2$ pairs
- Look at each par: if the two elements are different, discard both of them; if they are the same, keep just one of the,

Show that after this procedure there are at most $n/2$ elements left, and that they have a majority element if A does.)

Total points: 100