## Problem 1

We know that:

$$p(w) = p(w \mid b) \cdot p(b) + p(w \mid w) \cdot p(w)$$
$$p(b) = 1 - p(w)$$

so that, since we know $p(w \mid b)$ and $p(w \mid w)$, we have a linear system with two equations and two unknowns and can solve for $p(w) = \frac{3}{4}$, $p(b) = \frac{1}{4}$.

Plugging this into the formula for unconditional entropy gives:

$$\frac{1}{4}\log 4 + \frac{3}{4}\log\frac{4}{3} \approx 0.811278 \text{ bits}$$

and for conditional entropy

$$\frac{3}{4}(-.95\log .95 - .05\log .05) + \frac{1}{4}(-.15\log .15 - .85\log .85) \approx 0.367257 \text{ bits}$$

The ratio is $0.811278/0.367257$, for a factor of 2.2 times saving.

## Problem 2

We have $00011011010 = 0.1064453125$.

The initial intervals are partitioned by $[0, .1, .3, 1)$. The message falls in the interval $[.1, .3)$ so the first letter is **b**.

On the second round the intervals are [.1,.12,.16,.2) so the second letter is **a**.

On the third round the intervals are [.1,.102,.106,.12) so the third letter is **c**.

On the final round the intervals are [.106,.1074,.1102,.12) so the fourth letter is **a**.

Although the entropy of the message set is $H = 1.15678$, the actual information content of the four letters is $-(\log(.2) + \log(.1) + \log(.7) + \log(.1)) = 9.48$. This has an average of about 2.37 bits per letter, more than twice the per message entropy. The problem is the four letters which are sent are fairly unlikely under our initial model (especially the two **a**s), so the message sequence actually expanded.

## Problem 3

The idea is to use table lookup. Basically, create a table with $2^{w/2}$ entries. We can use the next $w/2$ bits of our message sequence as an integer index into the table. Each entry says what the first codeword of a string of length $w/2$ maps to and also the length of that codeword. We then shift over by the required number of bits. We can therefore decode each message and move our "cursor" to the next message in constant time.

Note the table can include many repeated entries. For example if $w/2 = 4$ and we have a codeword **01** for the letter **a** then the entries for **0100**, **0101**, **0110**, and **0111** would all contain the letter **a** and the length 2.

## Problem 4(a)

There are many ways to prove this. Here is one.

Define $n := \max_{w \in C} l(w)$. Let us construct a full binary tree of depth $n$ (i.e. leaves are at depth $n$, whereas the root is at depth 0) and let us mark leftward edges with 0 and rightward edges with 1, exactly as we did in the construction of Huffman code trees. Each node corresponds to a certain code specified by the path from the root to that node.

For each $w \in C$ let us define $\mathcal{L}(w)$ to be the set of all leaves that are descendants of the node that corresponds to the code $w$. Assume that for some codewords $w$, $v$ with $w \neq v$, at least one leaf belongs to both $\mathcal{L}(w)$ and $\mathcal{L}(v)$. Because this is a tree, one of $w, v$ is thus a descendant of the other. But our code was a prefix code, so this is impossible, and the sets $\mathcal{L}(w)$ and $\mathcal{L}(v)$ must be disjoint. The total number of leaves is $2^n$, so we have:

$$2^n \geq \sum_{w \in C} |\mathcal{L}(w)| = \sum_{w \in C} 2^{n-l(w)}.$$

Dividing through by $2^n$ yields the Kraft-McMillan inequality.

You could also show this property by structural induction on general (non-full) binary trees, or by starting a random process at the root of the tree that takes either path with equal probability and arguing that the total probability of hitting a leaf is no more than 1. (10 points)

## Problem 4(b)

We will prove this by contradiction. Assume that a prefix-free code $C$ has $2^k$ codewords: one shorter than $k$ bits, at most one longer than $k$ bits, and the rest at most $k$ bits long. Call the longest codeword $w$. Then

$$\sum_{(s,w') \in C} 2^{-l(w')} \geq 2^{1-k} + 2^{-l(w)} + (2^k - 2) \times 2^{-k} = 2^{1-k} + 2^{-l(w)} + 1 - 2^{1-k} = 1 + 2^{-l(w)} > 1.$$

This contradicts the Kraft-McMillan inequality, and thus such a code cannot exist. (10 points)

## Problem 5

The correct answers for your particular input, as well as example code, are now available on the assignment site: `http://realworld.herokuapp.com/compression`.

For part (d): the total number of bits does change if you reverse the order (as you can verify by rerunning your code on the reversed input). This is because trigrams looking “backwards” over English text are not exactly the same as trigrams looking “forwards,” and so our context dictionaries are all different, though the total number of bits shouldn’t be *too* far off.