Problem 1 – Original version

A. The escape count for context ac is 2. Thus the probability of occurence of a followed by context ac is $\frac{1}{4}$. The number of bits taken to encode a is $\log_2 4 = 2$.

B. Since the algorithm has not yet seen a d following context ac, we first output an escape character to move from $k = 2$ to $k = 1$. This takes $\log 2$ bits. Similarly, we output an escape character for the transition $k = 1$ to $k = 0$, using another $\log 3$ bits (corresponding to the escape count for context c). Finally, at level $k = 0$, we encode an escape character using $\log \frac{13}{3}$ bits, and then encode d using $\log 23$ bits. The total amount of information encoded is $\log 2 + \log 3 + \log (13/3) + \log 23 \approx 9.224$ bits.

The changes made to the dictionary are:

- Context ac : counts $a = 1; c = 1; d = 1; \$ = 3$.
- Context c : counts $a = 2; b = 2; d = 1; \$ = 3$.
- Context empty : counts $a = 4; b = 2; c = 4; d = 1; \$ = 4$.

C. If we use exclusion, for the context c, we already know from the previous step, that the following character is not a. Thus we can use an escape count of 1 and a total count of 3 instead of 2 and 6 respectively. This increases the probability of the next character beign a d at this stage, and so we need only $\log 2$ bits to encode the esacpe, as opposed to $\log 3$ for the previous part. Likewise, for $k = 1$, we know that the following character cannot be a, b or c, thus we do not need to encode the escape at all!

The total number of bits required is $\log 2 + \log 2 + \log 23 \approx 6.524$.

D. Encoding a requires $\log 3 = 1.585$ bits. Encoding d requires $\log 3 + \log 13/3 + \log 23 = 8.224$ bits.

Problem 1 – Modified version

A. The escape count for context ab is 1. Thus the probability of occurence of a followed by context ac is $\frac{1}{2}$. The number of bits taken to encode a is $\log_2 2 = 1$.

B. Since the algorithm has not yet seen a d following context ab, we first output an escape character to move from $k = 2$ to $k = 1$. This takes $\log 2$ bits. Similarly, we output an escape character for the transition $k = 1$ to $k = 0$, using another $\log 2$ bits (corresponding to the escape count for context c). Finally, at level $k = 0$, we encode an escape character using $\log \frac{13}{3}$ bits, and then encode d using $\log 23$ bits. The total amount of information encoded is $\log 2 + \log 2 + \log (13/3) + \log 23 \approx 8.639$ bits.

The changes made to the dictionary are:

- Context ab : counts $c = 1; d = 1; \$ = 2$.
- Context b : counts $c = 1; d = 1; \$ = 2$.
- Context empty : counts $a = 4; b = 2; c = 4; d = 1; \$ = 4$.

C. If we use exclusion, for the context b, we already know from the previous step, that the following character is not c. Thus we do not need to encode the escape at all! Likewise, for $k = 1$, we know that the following character cannot be c. Thus we can use an escape count of 2 and a total count of 8 instead of 3 and 13 respectively. This increases the probability of the next character being a d at this stage, and so we need only $\log 4$ bits to encode the esacpe, as opposed to $\log 13/3$ for the previous part.

   The total number of bits required is $\log 2 + 0 + \log 4 + \log 23 \approx 7.524$.

D. Encoding c requires $\log 2 = 1$ bit. Encoding d requires $\log 2 + \log 13/3 + \log 23 = 7.639$ bits.

Problem 2

A. LZ77 first encodes the triple $(0, 0, a)$ for the first character. Since the lookahead buffer is unbounded, the next match is $n - 1$ bits long (a^{n-1} starting at the first character matches a^{n-1} starting at the second character). Thus the next triple output by LZ77 is $(1, n - 2, a)$. This is the entire encoding of the string. The first triple roughly takes $2 + \log \frac{1}{p_a}$ bits to encode. The second triple takes 2 bits for the 1, $\log \frac{1}{p_a}$ for the character a and roughly $2 \log n$ bits to encode $n - 2$. Thus the total number of bits taken is $O(\log n + \log \frac{1}{p_a})$.

B. LZW also starts with the encoding of character a for the first two characters. At this point, it makes a new entry into the dictionary, corresponding to the substring aa. It encodes the next two characters using this new entry, and at the same time makes the entry aaa into the dictionary.

   Proceeding similarly, at the ith step, the algorithm has the entries $a, a^2, a^3, \cdots, a^i$ in the dictionary. The algorithm encodes the next i bits using the dictionary entry corresponding to a^i and makes a new entry for the substring a^{i+1}. The total number of entries made in this manner are given by k where k is a solution to the following equations:

$$1 + 2 + 3 + \cdots + k^2 \geq n$$

$$1 + 2 + 3 + \cdots + (k-1)^2 < n$$

   We get that $k = O(\sqrt{n})$. Assume that new entries in the dictionary start at the position c. Then, the i-th tuple output by the program encodes the position $i + c$ in the dictionary. Assuming that gamma codes are used, this takes roughly $O(\log \frac{1}{p_a} + \log (i + c))$ bits. Thus the total number of bits used is given by

$$\sum_{i=1}^{i=k} O(\log \frac{1}{p_a} + \log (i + c)) = O(\sqrt{n}(\log \frac{1}{p_a} + \log n))$$

   .

   Note that LZW takes almost a factor of $\sqrt{n}$ more bits than LZ77 to encode the same string. LZ77 with an unbounded lookahead buffer is known to produce an optimal encoding, but is impractical with respect to the time taken to encode the string.

Problem 3

A. For a character $c \in s$, and $w \in W^k$, let $p(c|w)$ denote the probability that some occurence of w in the string is followed by the character c. Also let $prec(c)$ denote the k-character substring preceding c in s.

Then, the kth order conditional entropy of s is given by $H_k(s) = -\sum_{c\in s} \log p(c|prec(c)) = -\sum_{w\in W^k} \sum_{c\in s:prec(c)=w} \log p(c|w) = -\sum_{w\in W^k} \sum_{c\in succ(w)} \log p(c|w)$.

But, $-\sum_{c\in succ(w)} \log p(c|w)$ is simply the total number of bits used to encode $succ(w)$ using single character probabilities. This is equal to $|succ(w)|H_0(succ(w))$ by the definition of H_0.

Thus we have $H_k(s) = \sum_{w\in W^k} |succ(w)|H_0(succ(w))$.

B. Since BW sorts strings from right to left, all strings containing some $w \in W^k$ as their suffix are adjacent in the sorted list. Consequently, all characters following some occurence of w in s appear together in the sorted list. For example, for $w = si$ in the string $mississippi$, the rotated string $sippimissi$ immediately precedes $pimississi$, and so the characters s and p appear consecutively in the sorted list.

C. Let us number the as and bs in order of their appearance in the string. Then, the substring corresponding to a_1 is $a(ba)^{20}bb$, that corresponding to a_2 is $(ba)^{20}bba$, while, that corresponding to a_i $(i > 2)$ is $(ba)^{22-i}bbaa(ba)^{i-3}b$. Similarly, the substring preceding b_i $(i \leq 20)$ is $a(ba)^{20-i}bbaa(ba)^{i-1}$, that corresponding to b_{21} is $baa(ba)^{20}$, and that corresponding to b_{22} is $aa(ba)^{20}b$.

Sorting these, we get the sequence $b_1b_2\ldots b_{21}a_2a_3\ldots a_{22}b_{22}a_1$, or $b^{21}a^{21}ba$.

The above string contains 22 as and 22 bs, so Arithmetic encoding assigns each character a probability of 0.5. Each character takes 1 bit to encode, resulting in an encoding of approximately 44 bits (ignoring the bits required to encode the actual characters).

D. MTF transforms the string $b^{21}a^{21}ba$ (obtained in the previous part) to $10^{20}10^{20}11$. The probability for character 0 is $\frac{40}{44}$, while that for character 1 is $\frac{4}{44}$. Thus the number of bits taken by Arithmetic Encoding to encode this string is roughly $40\log 1.1 + 4\log 11 \approx 20$.

E. Let k be any arbitrary number.

Let $W^k = \{w_1, \cdots, w_l\}$ be sorted in lexicographic order from right to left. From part (b) we know that for any $w_i \in W^k$, $\pi(succ(w_i))$ is a substring of s' for some permutation π. Since $\{w_i\}$ are sorted, part (b) implies that $s' = \pi_1(succ(w_1))\pi_2(succ(w_2))\cdots\pi_l(suc(w_l))$ for appropriate permutations $\pi_1, \cdots, \pi_l$.

Let $s_i = \pi_i(succ(w_i))$. From [Manzini'99], we know that the number of bits taken by MTF to encode s' is at most $c\sum_i |s_i|H_0(s_i)$. But, $|s_i| = |succ(w_i)|$, and, $H_0(s_i) = H_0(succ(w_i))$, because permutation does not effect the zeroth order entropy of a string. So we get that the number of bits taken to encode s' is at most

$$c\sum_i |succ(w_i)|H_0(succ(w_i)) = c\sum_{w\in W^k} |succ(w)|H_0(succ(w)) = cH_k(s)$$

where the last equality follows from part (a).

Problem 4

A. Recall that we can find the Harr transform by taking averages and differences iteratively. The coefficients corresponding to H_{ij} are given by differences computed on vector a at the $(k-i)$-th stage. The first few components of the vector a at the $(k-i)$-th stage contain averages of groups of 2^{k-i-1} elements of the original function. Thus the difference between two consecutive such elements is 2^{k-i-1}. So the coefficient of H_{ij} is $1/2 * 2^{k-i-1} = 2^{k-i-2}$.

Similarly, the constant component is simply the average of all the elements, which is equal to $\frac{1}{n}\frac{n(n+1)}{2} = \frac{n+1}{2}$.

So we get $f(x) = \frac{n+1}{2} + \sum_{i<k} \sum_{j \leq 2^i - 1} 2^{k-i-2} H_{ij}(x)$.

Similarly, solving for the second function, we get that $f(x) = \frac{3}{8} + \frac{3}{8}H_{00} + \frac{1}{4}H_{10} + \frac{1}{2}H_{21}$.

B. From the second equation, we get

$$a_{2i} = l_i - \lfloor \frac{h_{i-1} + h_i + 2}{4} \rfloor$$

Using this, we can compute all the even components. Then, we can compute the odd components, by using the following (derived from the first equation):

$$a_{2i+1} = h_i + \lfloor \frac{a_{2i} + a_{2i+1}}{2} \rfloor$$