1

Symbol	*Encoding*
A	0
B	01
C	10
D	1

With the variable-length binary code above, what is the string "001" an encoding of?

1. AB
2. CD
3. AAD
4. Not enough information to answer

(4) is the correct answer. This is example of why it is necessary to have prefix-free encoding, we can't tell what is the proper way to decode since multiple keys could have the same prefix.

2

Symbol	*Encoding*
A	0
B	10
C	110
D	111

Symbol	*Frequency*
A	60%
B	25%
C	10%
D	5%

What is the average number of bits per symbol used by the variable-length code above?

1. 1.5
2. 1.55
3. 2
4. 2.5

(2) is the correct answer $.1 \cdot 6 + .25 \cdot 2 + .1 \cdot 3 + .05 \cdot 3 = 1.55$.

3 How many mergers will Huffman's greedy algorithm perform before halting?

1. $n-1$
2. n
3. $\frac{(n+1)\cdot n}{2}$
4. Not enough information to answer

(1) is the correct answer. We start with n trees, where n is the number of alphabet symbols. Each merge replaces a pair of trees with a single tree and thus decrease the total number of trees by 1. This process will continue until one tree remains which is after $n-1$ iterations.

4 Suppose in the 1st iteration of algorithm we merge symbols a and b. what should be the frequency p_{ab} of this new symbol?

1. $max\{p_a, p_b\}$
2. $min\{p_a, p_b\}$
3. $p_a \cdot p_b$
4. $p_a + p_b$

(4) is the correct answer.

5 Prove that Huffman's algorithm computes a binary tree that minimizes the average encoding length that is $L(T) = \sum_{i\in\Sigma} p_i[depth\, of\, leaf\, i\, in\, T]$.

We proof by strong induction on the alphabet size $n = |\Sigma|$, where $n \geq 2$. When $n = 2$ the Huffman greedy algorithm outputs the optimal tree with 1 bit per symbol. We assume the hypothesis holds for all i less than n. Let $\Sigma' = \Sigma$ with the smallest frequencies a, b replaced by the meta-symbol ab. We define the frequency of $p_{ab} = p_a + p_b$. There is an bijective correspondence between T and T' where T is the tree for the Σ and T' is the tree for Σ'. For every pair T' and T,

$$\begin{aligned} L(T) - L(T') &= p_a \cdot [depth\, of\, a\, in\, T] + p_b \cdot [depth\, of\, b\, in\, T] - p_{ab} \cdot [depth\, of\, ab\, in\, T'] \\ &= p_a(d+1) + p_b \cdot (d+1) - (p_a + p_b) \cdot d \\ &= p_a + p_b \end{aligned}$$

Huffman's algorithm computes a tree $\hat{T}'$ that minimizes $L(T')$ for Σ'. As a lemma We prove that there is an optimal tree for Σ in X_{ab} where X_{ab} are the set of trees that have the symbols a and b as siblings. We do this by an exchange argument. Let $T^\star$ be any tree that minimizes the $L(T)$ for Σ. Let

x, y be siblings ath the deepest level of $T^\star$. we obtain $\hat{T}$ from $T^\star$ by swapping x with a and y with b.

$$\begin{aligned} L(T^\star) - L(\hat{T}) &= (p_x - p_a) \cdot [depth\, of\, x\, in\, T^\star - depth\, of\, a\, in\, T^\star] + (p_y - p_b) \cdot)[depth\, of\, y\, in\, T^\star - depth\, of\, a\, i \\ &\geq 0 \\ L(T^\star) &\geq L(\hat{T}) \end{aligned}$$

Hence Huffman will return an optimal tree for alphabet Σ.

6 Consider the following symbol frequencies for a five-symbol alphabet:

Symbol	*Frequency*
A	32%
B	25%
C	20%
D	18%
E	5%

What is the average encoding length of an optimal prefix-code?

1. 2.23
2. 2.4
3. 3
4. 3.45

(1) is the correct answer. The average encoding length of a tree T is given by the equation $L(T) = \sum_{i \in \Sigma} p_i \cdot [depth\, of\, leaf\, i\, in\, T]$. When we construct the huffman coding tree we get that A is encoded with 00, B is encoded with 01, C is encoded with 11, D is encoded with 100 and E is encoded with 101. Using the formula $L(T) = \sum_{i \in \{A,B,C,D,E\}} p_i \cdot [depth\, of\, leaf\, i\, in\, T] = .32 \cdot 2 + .25 \cdot 2 + .2 \cdot 2 + .18 \cdot 3 + .05 \cdot 3 = 2.23$.

7 Consider the following symbol frequencies for a five-symbol alphabet:

$Symbol$	$Frequency$
A	16%
B	8%
C	35%
D	7%
E	34%

What is the average encoding length of an optimal prefix-code?

1. 2.11
2. 2.31
3. 2.49
4. 2.5

(1) is the correct answer. The average encoding length of a tree T is given by the equation $L(T) = \sum_{i \in \Sigma} p_i \cdot [depth\, of\, leaf\, i\, in\, T]$. When we construct the Huffman encoding tree we obtain that A should be encoded with 010, B should be encoded with 0110, C should be encoded with 1, D should be encoded with 0111 and E should be encoded with 00. From this we use the equation to $L(T) = \sum_{i \in \{A,B,C,D,E\}} p_i \cdot [depth\, of\, leaf\, i\, in\, T] = 3 \cdot .16 + 4 \cdot .08 + 1 \cdot .35 + 4 \cdot .07 + 2 \cdot .35 = 2.11$

8 What is the maximum numbef of bits that Huffman's greedy algorithm might use to encode a single symbol?

1. $log_2 n$
2. lnn
3. $n - 1$
4. n

(3) is the correct answer. We could imagine a leaf A' such that it is involved in every merger in the algorithm. By (3) there are at most $n - 1$ iterations and a merger increases the depth of trees by 1 hence the maximum would be $n - 1$.

9 Which of the following statements about Huffman's greedy algorithm are true?

1. A letter with frequency at least 0.4 will never be encoded with two or more bits

2. A letter with frequency at least 0.5 will never be encoded with two or more bits

3. If all symbols frequencies are less than 0.33, all symbols will encoded with at least two bits

4. If al symbols frequencies are less than 0.5, all symbols will be encoded with at least two bits.

(3) and (4) are correct.

10 Given an implementation of Huffman's greedy algorithm that uses a single invocation of a sorting subroutine, followed by a linear amount of additional work.

We first sort the symbols by their frequency and insert them into a queue in increasing order, we label this queue q_1. We'll keep a second queue q_2 that'll keep the trees that have been merged and have them ordered in increasing order. In the first iteration we pop the first two elements labeled a and b from q_1 and add them into the q_2 with the frequency of the a and b being added. In every other iteration greater than 1 we peek at both queues and pop the smaller element from one of the queues. If the queue that evicted a member still has elements in it we peek and see if its frequency is smaller than the other element in the front of the other queue, if so we pop that element. We then merge those elements and insert it into q_2. This works exactly like the Huffman greedy algorithm. The algorithm always picks the two smallest trees merges them and places them into a bigger tree.