Homework 4:

Due: November 2nd, 2025 at 2:30p.m.

This homework must be typed in LaTeX and submitted via Gradescope.

Please ensure that your solutions are complete, concise, and communicated clearly. Use full sentences and plan your presentation before your write. Except where indicated, consider every problem as asking for a proof.

Problem 1. Suppose the symbols a, b, c, d, e occur with frequencies 1/2, 1/4, 1/8, 1/16, 1/16, respectively.
(a) What is the Huffman encoding of the alphabet?
(b) If this encoding is applied to a file consisting of 1,000,000 characters with the given frequencies, what is the length of the encoded file in bits?

Solution. **(a) Building the Huffman tree.** Merge the two least frequent symbols repeatedly:

$$d(1/16),\, e(1/16) \to (1/8); \quad c(1/8),\, (1/8) \to (1/4); \quad b(1/4),\, (1/4) \to (1/2); \quad a(1/2),\, (1/2) \to 1.$$

Thus the codeword lengths are $\ell(a) = 1,\ \ell(b) = 2,\ \ell(c) = 3,\ \ell(d) = \ell(e) = 4$. One valid canonical assignment is:

$$a \mapsto 0, \quad b \mapsto 10, \quad c \mapsto 110, \quad d \mapsto 1110, \quad e \mapsto 1111.$$

(Any left/right tie-breaking yields an equivalent optimal code up to renaming the bits.)

(b) Expected length. The expected bits per symbol is

$$\mathbf{E}[\ell] = \tfrac{1}{2} \cdot 1 + \tfrac{1}{4} \cdot 2 + \tfrac{1}{8} \cdot 3 + \tfrac{1}{16} \cdot 4 + \tfrac{1}{16} \cdot 4 = 0.5 + 0.5 + 0.375 + 0.25 + 0.25 = 1.875.$$

For 1,000,000 characters the encoded length is $1.875 \times 10^6 = 1{,}875{,}000$ bits. □

Problem 2. We use Huffman's algorithm to obtain an encoding of alphabet $\{a, b, c\}$ with frequencies f_a, f_b, f_c. In each of the following cases, either give an example of frequencies (f_a, f_b, f_c) that would yield the specified code, or explain why the code cannot possibly be obtained (no matter what the frequencies are).

(a) Code: {0, 10, 11}

(b) Code: {0, 1, 00}

(c) Code: {10, 01, 00}

Solution. **(a) Possible.** This has lengths $(1, 2, 2)$, which is exactly what Huffman produces on three symbols: combine the two smaller to depth 2, and the largest gets depth 1. Any frequencies with $f_a \geq \max\{f_b, f_c\}$ work; e.g. $(f_a, f_b, f_c) = (0.6, 0.25, 0.15)$ yields $\{a \to 0,\ b \to 10,\ c \to 11\}$.

(b) Impossible. The set $\{0, 1, 00\}$ is not prefix-free because 0 is a prefix of 00. No prefix code (Huffman or otherwise) can assign these codewords.

(c) Impossible. The lengths are $(2, 2, 2)$ (all equal). While a prefix code with these words exists (Kraft sum $= 3/4$), Huffman on three symbols always yields lengths $(1, 2, 2)$: it first merges the two least frequent symbols, then merges that pair with the remaining symbol, making one codeword of length 1 and two of length 2 (ties do not change this structure). Hence $\{10, 01, 00\}$ cannot arise from Huffman for any frequencies. □

Problem 3. Given an n-bit binary integer, design a divide-and-conquer algorithm to convert it into its decimal representation. For simplicity, you may assume that n is a power of 2.

1. Provide a succinct (but clear) description of your algorithm, including pseudocode.
2. Prove the correctness of your algorithm.
3. Analyze the running time of your algorithm. Assume that it is possible to multiply two decimal integers numbers with at most m digits in $O(m^{\log_2 3})$ time.

Hint: An n-bit binary integer x can be expressed as $x = (x_{n-1}, x_{n-2}, \cdots, x_1, x_0)_2$ where $x_i \in \{0, 1\}$. Let $x_\ell = (x_{n/2-1}, x_{n/2-2}, \cdots, x_1, x_0)_2$ be the $(n/2)$-bit binary integer corresponding to the $(n/2)$ least significant digits of x. Let $x_m = (x_{n-1}, x_{n-2}, \cdots, x_{n/2+1}, x_{n/2})_2$ be the $(n/2)$-bit binary integer representing the $(n/2)$ most significant digits of x. Then, $x = x_\ell + 2^{n/2} \cdot x_m$. This should suggest us a way to set up a divide and conquer strategy... :) Careful about the number of subproblems!

Solution. 1. Using the hint, the idea is to split the n-bit integer into the first half and second half: call these $n/2$-bit halves x and y, respectively. Then, we want to compute $2^{n/2}x + y$. Continue calling the algorithm on x, y until they are of 1-bit each, at which point we return the value itself.

2. Clearly the base cases of length 1 work, since $0_2 = 0$ and $1_2 = 1$. It suffices to show that $x = 2^{n/2}x_l + x_r$ is correct, where x_l, x_r are as defined in the hint. Indeed, notice that

$$\begin{aligned} x &= (x_n, x_{n-1}, \cdots, x_1)_2 \\ &= (x_n, \cdots, x_{n/2+1}, 0, \cdots, 0)_2 + (x_{n/2}, \cdots, x_1)_2 \\ &= (x_n, \cdots, x_{n/2} + 1, 0, \cdots, 0)_2 + x_r \\ &= 2^{n/2}(x_n, \cdots, x_{n/2+1})_2 + x_r \\ &= 2^{n/2}x_l + x_r \end{aligned}$$

since appending a zero to the end of a binary integer is equivalent to multiplying by 2 in decimal and there are $n/2$ zeros. Thus, the algorithm properly handles base cases and correctly combines the results from splitting.

3. Let $T(n)$ denote the number of operations needed for an $n-$bit binary integer. I claim that

$$T(n) = 2T(n/2) + O(n^{\log_2 3})$$

After splitting, the conversion of x_l and x_l into decimal clearly take $T(n/2)$ each, yielding the $2T(n/2)$ term. As for the combine step, it suffices to determine the runtime of multiplying $2^{n/2}$ by x_l, since addition is done in linear time, $O(n)$.

To compute $2^{n/2}$, we can, say, repeatedly square starting at 2. This requires squaring $\log_2(n/2) = O(\log n)$ times. Squaring is at worst multiplying two $n/4$-bit integers (in decimal). In decimal, we have $\log_{10}(2^{n/4}) = n/4 \cdot \log_{10}(2) = O(n)$ digits, so multiplication takes

$O(n^{\log_2 3})$ time. We do this for $n/8$, $n/16$, etc, so the runtime is

$$\begin{aligned} O(n^{\log_2 3} + (n/2)^{\log_2 3} + \cdots + 1) &= O(n^{\log_2 3} + \frac{1}{3}n^{\log_2 3} + \frac{1}{9}n^{\log_2 3} + \cdots) \\ &= O\left(\frac{1}{1-1/3}n^{\log_2 3}\right) \\ &= O(n^{\log_2 3}) \end{aligned}$$

It remains to multiply $2^{n/2}$ and x_l. However, both are $n/2$-bit integers, meaning the number of digits in the decimal representation of x_l and $2^{n/2}$ is

$$O(\log_{10}(2^{n/2})) = O(n)$$

Multiplying two decimal integers with at most m digits takes $O(m^{\log_2 3})$ time, and since each of x_l and $2^{n/2}$ have at most $O(n)$ digits, the multiplication takes $O(n^{\log_2 3})$ time. The recurrence relation becomes

$$T(n) = 2T(n/2) + O(n^{\log_2 3}) + O(n^{\log_2 3}) = 2T(n/2) + O(n^{\log_2 3})$$

We apply the Master theorem. Since $\log_2(3) > \log_2(2) = 1$, we have an instance of Case 3. Indeed, setting $\delta = 2/3$,

$$\delta n^{\log_2 3} = \frac{2}{3}n^{\log_2 3} = 2\frac{n^{\log_2 3}}{2^{\log_2 3}} = 2(n/2)^{\log_2 3} = 2f(n/2)$$

as desired. By Master theorem, then, $T(n) = \Theta(n^{\log_2 3})$, which is our overall runtime.

□