CS 260 - Assignment 17

Professor Mark W. Boady

Week 10

1 Introduction

This worksheet is worth 100 points.
You may complete this worksheet using any of the following methods.

- Print out the PDF. Complete by hand. Scan the file.
- Write directly on the PDF with editing software
- Write the Answer in MS Word, Notepad, etc. You **do not** have to rewrite the questions. Just clearly label each answer in your file.

2 Assignment 17

Question 1 : 4 points

Below is a string of text.

```
1 efdefefdebcdfdebfeccedfaffffaadcf
2 dbfcffcefdeececbdecdefdfebcadbafda
3 deeceebdefedfddddfcffadefdbbffeef
```

The percentage for each character is given below.

Letter	Percent
a	7
b	9
c	12
d	22
e	23
f	27

Assume each character is stored as an 8-bit ASCII value. How many bits would it take to store the text?

Question 2 : 4 points

The Huffman Compression Algorithm uses a forest of trees. Draw a single node for each letter. Each node will contain the letter and the percentage you computed in question 1.

Question 3 : 4 points

Take the two nodes with the smallest percents and combine them into a new tree.

The parent will contain the total percent of both children. The smaller percent will be the left child.

Draw the new forest below.

Question 4 : 4 points

Take the two nodes with the smallest percents and combine them into a new tree.

Hint: It is useful to keep the nodes sorted when drawing the forest.

Question 5 : 4 points

Take the two nodes with the smallest percents and combine them into a new tree.

Hint: Just take the two smallest percents. You may need multiple trees!

Question 6 : 4 points

Take the two nodes with the smallest percents and combine them into a new tree.

Question 7 : 4 points

Take the two nodes with the smallest percents and combine them into a new tree.

Question 8 : 4 points

Redraw your tree from the previous question.

Add binary Encodings.

For every left edge, put a 0 on the edge.

For every right edge, put a 1 on the edge.

Question 9 : 6 points

For each letter, write the compressed binary code you created next to it.

The binary code of a letter is the value of the edges from the root to that letter's node.

Question 10 : 3 points

Determine how many bits are needed to store the string (from Question 1) using a Huffman encoding.

Question 11 : 0 points

3] What is your Compression Ratio? (Uncompressed Size) / (Compressed Size)

Question 12 : 8 points

The run-time of recursive algorithms can be described using recursive formula.

For Binary Search: Compare to the middle element and (worst-case) search half the remaining list.

Let the function $T(n)$ give the running time of Binary Search on a sorted array with n elements.

If the array has 1 element, it takes 1 comparison to find the answer. That means $T(1) = 1$.

If the array has $n > 1$ elements, then we do one comparison and search half the remaining elements. That means $T(n) = 1 + T(\frac{n}{2})$.

The full expression for Binary Search is

$$T(n) = \begin{cases} 1 & n = 1 \\ 1 + T\left(\frac{n}{2}\right) & n > 1 \end{cases} \tag{1}$$

Here are some example evaluations. We will only look at cases where n is a power of two (for easy math).

$$T(2) = 1 + T\left(\frac{2}{2}\right) = 1 + T(1) = 1 + 1 = 2 \tag{2}$$

$$T(4) = 1 + T\left(\frac{4}{2}\right) = 1 + T(2) = 1 + 2 = 3 \tag{3}$$

$$T(8) = 1 + T\left(\frac{8}{2}\right) = 1 + T(4) = 1 + 3 = 4 \tag{4}$$

For **Merge Sort**, we sort two halves of the list and then merge them. The recursive formula is.

$$M(n) = \begin{cases} 1 & n \leq 1 \\ 2M\left(\frac{n}{2}\right) + n & n > 1 \end{cases} \tag{5}$$

(a) (2 points) Compute $M(1)$

(b) (2 points) Compute $M(2)$

(c) (2 points) Compute $M(4)$

(d) (2 points) Compute $M(8)$

Question 13 : 5 points

The process from Question 1 find a numeric answer, but doesn't tell up much about the pattern. It is better to analyze the function algebraically.

Let n be some large number. What happens for $T(n)$? We want to find a formula in terms of the iteration k of the recursion.

$$T(n) = T(\frac{n}{2}) + 1 \quad \text{Iteration 1} \tag{6}$$

$$= [T(n/4) + 1] + 1 = T(n/4) + 2 \quad \text{Iteration 2} \tag{7}$$

$$= [T(n/8) + 1] + 2 = T(n/8) + 3 \quad \text{Iteration 3} \tag{8}$$

$$= T(n/2^k) + k \quad \text{Iteration k} \tag{9}$$

Expand $M(n)$ until you see a pattern and write it in terms of the iteration k.

Question 14 : 3 points

We came up with the expression $T(n) = T(n/2^k) + k$ for the k-th iteration. How many iterations will the algorithm run?

We know it stops at $T(1)$, so we solve $T(1) = T(n/2^k)$ for k.

$$1 = n/2^k \tag{10}$$
$$2^k = n \tag{11}$$
$$k = \log_2(n) \tag{12}$$

You should have found an expression in terms of $M(n/2^k)$ in the previous question.

Solve for k in terms of n.

Question 15 : 5 points

Now that we know k, we can plug it into our formula. This will give us a closed form expression!

$$T(n) = T(n/2^k) + k \tag{13}$$
$$= T\left(n/2^{log_2(n)}\right) + \log_2(n) \tag{14}$$
$$= T(1) + \log_2(n) \tag{15}$$
$$= 1 + \log_2(n) \tag{16}$$

Plug k into your expression for $M(n)$

Question 16 : 20 points

The above method **always** works in theory. If it can actually be done algebraically is another question.

In most cases, we only care that $T(n) = \Theta(\log_2(n))$. The **MASTER THEOREM** gives us a short cut for any recursive expression of the form $T(n) = aT(n/b) + f(n)$. (Note: T is normally used for any run-**t**ime function.)

The first step is to break $T(n)$ up into a, b, and $f(n)$. For example, $T(n) = 8T(n/2) + 5n^2 + 4$ breaks up into $a = 8$, $b = 2$, and $f(n) = 5n^2 + 4$. The function $f(n)$ cannot be recursive.

Give a, b, and $f(n)$ for each of the below.

(a) (2 points) $T(n) = 9T(n/3) + 4n$

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

(c) (2 points) $T(n) = 4T(n/2) + n^3$

(d) (2 points) $T(n) = 25T(n/5) + 10n$

(e) (2 points) $T(n) = T(n/2) + n$

For each computation, we will need $c = \log_b(a)$. Compute c for each expression.

(a) (2 points) $T(n) = 9T(n/3) + 4n$

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

(c) (2 points) $T(n) = 4T(n/2) + n^3$

(d) (2 points) $T(n) = 25T(n/5) + 10n$

(e) (2 points) $T(n) = T(n/2) + n$

Question 17 : 10 points

The Master Theorem has three cases. We will look at two now.

Partial Master Theorem:

Let

$$T(n) = \begin{cases} 1 & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases} c = \log_b(a)$$

- Case 1: **IF** $f(n) = O\left(n^{c-\epsilon}\right)$ for some constant $\epsilon > 0$ **THEN** $T(n) = \Theta(n^c)$
- Case 2: Coming Soon
- Case 3: **IF** $f(n) = \Omega\left(n^{c+\epsilon}\right)$ for some constant $\epsilon > 0$ and $af(n/b) < xf(n)$ for $x < 1$ and large n **THEN** $T(n) = \Theta(f(n))$

Note: For today's lab, we will not worry about the condition $af(n/b) < xf(n)$ for $x < 1$ and large n.

Our Example was: $T(n) = 8T(n/2) + 5n^2 + 4$ breaks up into $a = 8$, $b = 2$, and $f(n) = 5n^2 + 4$

That means $c = \log_2(8) = 3$. We now compare n^3 with $f(n) = 5n^2 + 4$.

We find that $5n^2 + 4 = O(n^{3-\epsilon})$ where $\epsilon = 1$. In this case ϵ is equal to 1 because a tighter bound would be $5n^2 + 4 = O(n^2)$.

We can draw the conclusion that **Case 1** selected. That means $T(n) = \Theta(n^3)$.

Determine the Theta value of each of the below expressions. Only Case 1 or Case 3 are used in these questions.

(a) (2 points) $T(n) = 9T(n/3) + 4n$

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

(c) (2 points) $T(n) = 4T(n/2) + n^3$

(d) (2 points) $T(n) = 25T(n/5) + 10n$

(e) (2 points) $T(n) = T(n/2) + n$

Question 18 : 8 points

Cases 1 and 3 of the master theorem are fairly easy to determine because the largest exponent of n^c and $f(n)$ are different. What happens when they are the same? This is when **Case 2** is needed.

Master Theorem:

Let

$$T(n) = \begin{cases} 1 & n = 1 \\ aT(n/b) + f(n) & n > 1 \end{cases} c = \log_b(a)$$

- Case 1: **IF** $f(n) = O\left(n^{c-\epsilon}\right)$ for some constant $\epsilon > 0$ **THEN** $T(n) = \Theta(n^c)$
- Case 2: **IF** $f(n) = \Theta\left(n^c \log^k(n)\right)$ for $k \geq 0$ **THEN** $T(n) = \Theta\left(n^c \log^{k+1} n\right)$.
- Case 3: **IF** $f(n) = \Omega\left(n^{c+\epsilon}\right)$ for some constant $\epsilon > 0$ and $af(n/b) < xf(n)$ for $x < 1$ and large n **THEN** $T(n) = \Theta(f(n))$

Let us look at two examples.

Example 1

$T(n) = 3T(n/3) + n$. We have $a = 3$, $b = 3$, and $f(n) = n$. We compute $c = \log_3(3) = 1$. We compare $f(n) = \Theta(n^c)$ because $n = \Theta(n^1)$. This falls into case 2. The value of $k = 0$ because $\log^0(n) = 1$. We conclude that $T(n) = \Theta(n \log n)$.

Example 2

$T(n) = 4T(n/2) + n^2 \log_n$. We have $a = 4$, $b = 2$, $f(n) = n^2 \log_n$. We compute $c = \log_2(4) = 2$. We compare $f(n) = \Theta(n^c \log n)$ because $n^2 \log n = \Theta(n^2 \log n)$. This falls into case 2. The value of $k = 1$. We conclude that $T(n) = \Theta(n^2 \log^2 n)$.

Both the following fall into Case 2. Determine the Theta expression for each.

(a) (4 points) $25T(n/5) + n^2 \log_2^3(n)$

(b) (4 points) $81T(n/9) + n^2$