COMP S265F Design and Analysis of Algorithms
Lab 5: Huffman Codes and the Master Theorem – Suggested Solution

Question 1.

(a) In this case, we have $a = b = 2$ and thus

$$\log_b a = \log_2 2 = 1 \; .$$

On the other hand, we have $d = 0$ because

$$O(1) = O(n^0)$$

Therefore,

$$d = 0 < 1 = \log_b a \; .$$

By the Master Theorem, $T(n) = O(n^{\log_b a}) = O(n)$.

(b) In this case, we have $a = b = 2$ and thus

$$\log_b a = \log_2 2 = 1 \; .$$

On the other hand, we have $d = 1$ because

$$O(n) = O(n^1)$$

Therefore,

$$d = 1 = \log_b a \; .$$

By the Master Theorem, $T(n) = O(n^{\log_b a} \log n) = O(n \log n)$.

Question 2. We can substitute the upper bound $T(n) \le cn - d$ to the formula:

$$\begin{aligned} T(n) &= T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 1 \\ &\le \left(c\left\lceil \frac{n}{2} \right\rceil - d\right) + \left(c\left\lfloor \frac{n}{2} \right\rfloor - d\right) + 1 \\ &= cn - 2d + 1 \\ &\le cn - d \qquad \text{(when } d \ge 1\text{)} \end{aligned}$$

Therefore, $T(n) \le cn - d$ and we can also conclude that $T(n) = O(n)$.

Question 3.

(a) We can move n disks from pole a to pole c using pole b as a buffer, as follows:

```
procedure HANOI(a, b, c, n)
    if n = 1 then
        Move the top disk from a to c
    else
        HANOI(a, c, b, n − 1)            ▷ Move the top n − 1 disks from a to b
        Move the top disk from a to c    ▷ Move the largest disk from a to c
        HANOI(b, a, c, n − 1)            ▷ Move the top n − 1 disks from b to c
    end if
end procedure
```

(b) Let $T(n)$ be the number of moves for moving n disks in our algorithm.
If $n = 1$, a single move is needed in line 3 of the algorithm.
If $n > 1$, the algorithm divide it into three subproblems:

1. Moving the top $n-1$ disks from the source a to the buffer b by a recursive call (line 5).
2. Make a single move of the largest disk from the source a to the destination c (line 6).
3. Moving the top $n-1$ disks from the buffer b to the destination c by a recursive call (line 7).

Therefore,

$$T(n) = \begin{cases} 2T(n-1)+1 & \text{if } n > 1; \\ 1 & \text{if } n = 1. \end{cases}$$

We can solve $T(n)$, as follows:

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2^2T(n-2)+2+1 \\ &= 2^3T(n-3)+2^2+2+1 \\ &= \cdots \\ &= 2^{n-1}T(1)+2^{n-2}+2^{n-3}+\cdots+2+1 \\ &= 2^{n-1}\cdot 1+\frac{2^{n-1}-1}{2-1} \\ &= 2^{n-1}+2^{n-1}-1 \\ &= 2^n-1\ . \end{aligned}$$

(c) We prove that our algorithm gives the solution with minimum number of moves by mathematical induction. Let $Opt(n)$ be the minimum number of moves for n disks in the optimal solution.

Base case. When $n = 1$, we must move one disk from the source pole A to the destination pole C, so $Opt(n) = 1 = T(n)$. Hence, our algorithm is optimal when $n = 1$.

Induction hypothesis. Suppose that our algorithm is optimal for moving $n-1$ disks, i.e., $Opt(n-1) = T(n-1)$.

Inductive step. Consider that there are n disks to be moved from the source pole A to the destination pole C.

If we want to move all the n disks from A to C, we must obey the rule that all the disks are arranged in decreasing order of sizes. In the optimal solution, the n-th disk from the top at A (i.e., the largest disk) must finally become the n-th disk from the top at C. The only way is to move the top $n-1$ disks from A to the buffer pole B, then move that largest disk from A to C, and finally move the top $n-1$ disks from the buffer pole to the destination pole.

Therefore, the optimal solution must move the top $n-1$ disks at least twice and then move the largest disk once, i.e., $Opt(n) \geq 2Opt(n-1)+1$.

By the induction hypothesis, $Opt(n) \geq 2T(n-1)+1 = T(n)$. Thus, our algorithm does not use more moves than the optimal algorithm, so our algorithm uses the minimum number of moves and $T(n) = Opt(n)$.