CSC301 HW3

Alex Zhang

Jan 2023

Question 1

(a)

Since $n \geq n-1 \geq n-2 \geq n-3 \geq \dots$, so that $n \cdot n \geq n \cdot (n-1)$. We can then apply this inequality with more numbers which

$$n \cdot (n-1) \cdot (n-2) \cdot (n-3) \dots 1 \leq n \cdot n \dots n$$

This inequality holds true because each element on the left side is smaller or equal to elements on the right side. Simplifying the inequality,

$$n! \leq n^n$$

which shows that it is true.■

(b)

Takes the $\log_{n/2}$ for $(n/2)^{n/2}$, which equals

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

Takes the $\log_{n/2}$ for n factorial. This equals

$$\log_{n/2}(n!) = \sum_{i=0}^{n-1} \log_{n/2}(n-i)$$

Given a log function $\log_a b$, as long as $b \geq a$, $\log_a b \geq 1$. Expanding $\sum_{i=0}^{n-1} \log_{n/2}(n-i)$:

$$\sum_{i=0}^{n-1} \log_{n/2}(n-i) = \log_{n/2}(n) + \log_{n/2}(n-1) + \dots + \log_{n/2} 1$$

We can get that all elements before $\log_{n/2}(n/2-1)$ is larger or equal to 1, and there are total $n/2+1$ elements before $n/2-1$ in this summation. Therefore, we can obtain the following inequality:

$$\sum_{i=0}^{n-1} \log_{n/2}(n-i) = \log_{n/2}(n) + \log_{n/2}(n-1) + \dots + \log_{n/2} 1 \geq n/2+1$$

Which is the same as,

$$\log_{n/2}(n!) \geq n/2+1 \geq n/2 = \log_{n/2}(n/2)^{n/2}$$

Exponentiates both sides,

$$n! \geq (n/2)^{n/2}$$

Just as the prompt.■

(c)

From question (a) and (b), we can get the inequality,

$$n^n \geq n! \geq (n/2)^{n/2}$$

Takes the log for all of them,

$$n \log n \geq \log(n!) \geq (n/2)\log(n/2)$$

Case 1: Big-Oh
Let $f(n) = \log(n!)$ and $c \cdot g(n) = c \cdot n \log n$. By definition, Since

$$\log(n!) \leq n \log n$$

We can let $c = 1$ and $N = 1$, and plug in the number into inequality,

$$f(n) = \log(n!) \leq n \log n = g(n)$$

for all $n \geq N$. Therefore,

$$\log(n!) = O(n \log n)$$

Case 2: Big-Omega
Since $\log(n!) \geq (n/2)\log(n/2)$, we can do some transformation on the right hand side,

$$\log(n!) \geq (n/2)\log n - (n/2)\log 2$$

When $n \geq 4$, $n/4 \log n \geq n/2$ and substitudes $n/2 \log 2$ with $n/4 \log n$, we can get:

$$\log(n!) \geq (n/2)\log n - n/4 \log n = n/4 \log n \text{ when } n \geq 4$$

By definition, let $f(n) = \log(n!)$, and $c \cdot g(n) = c \cdot n/4 \log n$. We can assume that for $c = 4$ and $N = 4$, the inequality

$$\log(n!) \geq n \log n$$

holds.
So for all $n \geq N$, then

$$\log(n!) = \Omega(n \log n)$$

Overall, if $\log(n!) = O(n \log n)$, and $\log(n!) = \Omega(n \log n)$, then

$$\log(n!) = \Theta(n \log n)$$

■

Question 2

Assume that the time complexity of function MULTIPLY for n-bits number is $T(n)$. In each recursion calls, the input is seperated into 3 parts(line 4 and 5), and following 5 times of recursions (showed in line 6 to 10). For partitioning, the time complexity is $O(n)$ since we assume the division by 3 is linear and the multiplication of 2 is just adding zeros behind. For the rest part, since number addtion and subtraction all cost linear time and division by 3 is also $O(n)$. We can say that for each recursion call, the time complexity is $O(n)$ and we can draw the following formula,

$$T(n) = 5T(n/3) + O(n)$$

Also the base case $T(1)$ when $n = 1$ also has time complexity $O(1)$ becuase multiplying two 1-bit number costs in constant time. Based on the information above, we are able to use master theorem to get the time complexity for MULTIPLY since $a = 5$, $b = 3$, and $d = 1$.
Using master theorem, we get

$$\log_b a = \log_3 5 \approx 1.46 > 1 = d$$

So we will apply third case for master theorem which

$$T(n) = O(n^{\log_b a}) = O(n^{\log_3 5}) = O(n^{1.46})$$

The time complexity for MULTIPLY is $O(n^{1.46})$.

Question 3

(a) $T(n) = T(n/2) + O(\log n)$

We can use back-substitution to solve the recursion.

$$\begin{aligned} T(n) &= T(n/2) + O(\log n) \\ T(n/2) &= T(n/4) + O(\log n) \\ T(n/4) &= T(n/8) + O(\log n) \\ &\vdots \\ T(1) &= O(1) \end{aligned}$$

Where for a number k,

$$T(n) = T(n/2^k) + k \cdot c \log n$$

In back-substitution, there are total L level which $n/2^L = 1 \Rightarrow L = \log_2 n$. Then,

$$\begin{aligned} T(n) &= T(1) + \log n \cdot c' \log n \\ T(n) &= c'(\log n)^2 + c \end{aligned}$$

Because $c'(\log n)^2$ is the dominating term in this equation and c is a constant , we can get the time complexity for $T(n) = O((\log n)^2)$

(b) $T(n) = 7 \cdot T(n/2) + O(n^2)$

In this case, since $a = 7$, $b = 2$, and $d = 2$, and $T(1) = O(1)$ we can use master theorem to solve the recurrence.

$$\log_b a = \log_2 7 \approx 2.807 > 2 = d$$

We will go in case 3 of master theorem then,

$$T(n) = O(n^{\log_b a}) = O(n^{2.807})$$

(c) $T(n) = T(n-1) + O(n)$

Back-substitution can be used for solving,

$$\begin{aligned} T(n) &= T(n-1) + O(n) \\ T(n-1) &= T(n-2) + O(n) \\ T(n-2) &= T(n-3) + O(n) \\ &\vdots \\ T(1) &= O(1) \end{aligned}$$

For a number k, $T(n)$ can be expressed as

$$T(n) = T(n-k) + c' \cdot kn$$

For leave level L, L satisfies $n - L = 1 \Rightarrow L = n - 1$ and plug in $k = L$,

$$\begin{aligned} T(n) &= T(1) + c' \cdot Ln \\ T(n) &= c \cdot 1 + c' \cdot (n-1)n \\ T(n) &= c + c'n^2 - c'n \end{aligned}$$

Since $c'n^2$ is the dominating term and c is a constant, we can get the time complexity $T(n) = O(n^2)$.

(d) $T(n) = 2 \cdot T(n/2) + O(n)$

Since $a = 2$, $b = 2$, and $d = 1$ with base case $T(1) = O(1)$, we can use master theorem solving the recurrence.

$$\log_b a = \log_2 2 = 1 = d$$

It goes into second case, where $T(n) = O(n^d \log n)$ and plugging in numbers.

$$T(n) = O(n \log n)$$

(e) $T(n) = 2 \cdot T(n-1) + O(1)$

We can still use the substitution method.

$$\begin{aligned} T(n) &= 2 \cdot T(n-1) + O(1) \\ T(n-1) &= 2 \cdot T(n-2) + O(1) \\ T(n-2) &= 2 \cdot T(n-3) + O(1) \\ &\vdots \\ T(1) &= O(1) \end{aligned}$$

Given a number k, uses $n - k$ to represent $T(n)$ which,

$$T(n) = 2^k \cdot T(n-k) + c \cdot \sum_{i=0}^{k-1} 2^i$$

In this case at leave level L,

$$T(n) = 2^L \cdot T(n-L) + c \cdot \sum_{i=0}^{L-1} 2^i$$

which $n - L = 1$, $L = n - 1$,

$$\begin{aligned} T(n) &= 2^{n-1} T(1) + c \cdot \sum_{i=0}^{n-2} 2^i \\ T(n) &= 2^{n-1} \cdot c' + c \cdot \frac{2^{n-1} - 1}{1} \\ T(n) &= 2^{n-1} \cdot (c' + c) + c \\ T(n) &= \frac{c' + c}{2} 2^n + c \end{aligned}$$

In this case since c' and c are both constant, the time complexity for $T(n)$ will be

$$T(n) = O(2^n)$$