Assignment #1

Bryan Estrada Chiang

1/18/2026

A A Complex Complexity Problem

1. A polylogarithmic function is any function whose upper bound function is $O(\log^k x)$ for $k > 1$. From the functions Yihan found, $\log^2 n$, $\log n^2$, $\ln \ln n$, 10000, $\log \log n$, $\sqrt{\log n}$, and $\log \sqrt{n}$ are polylogarithmic.

2. A sublinear function is a function $f(n)$ where its upper bound is $o(n)$, therefore $f(n) = o(n)$. From Yihan's function pool, $\log^2 n$, $\log n^2$,$\ln \ln n$, 10000, $\sqrt{n}$, $\log \log n$, $\sqrt{\log n}$, $\log \sqrt{n}$, and $\frac{n}{\log n}$ are sublinear.

3. A superlinear function is a function $f(n)$ whose lower bound is $\omega(n)$ such that $f(n) = \omega(n)$. From Yihan's function pool, 3^n, $n!$, $n^2 + n$, n^3, 2^{n+1}, $(n+1)!$, 2^n, $n \log n$, $5n^2 - 13n + 6$ are superlinear.

4. (a) Let $f(n) = \log(\log n)$ and $g(n) = \sqrt{\log n}$. Now let's analyze what function grows faster as $n \to \infty$.

$$\begin{aligned} \lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{\log(\log n)}{\sqrt{\log n}} \\ &= \frac{\log(\log \infty)}{\sqrt{\log \infty}} \\ &= \frac{\infty}{\infty} \end{aligned}$$

Because we obtained indeterminate form when plugging in ∞. Then, we can apply L'hopital rule.

$$
\begin{aligned}
\lim_{n\to\infty}\frac{f(n)}{g(n)} &= \lim_{n\to\infty}\frac{\log(\log n)}{\sqrt{\log n}} \\
&\overset{\text{L'H}}{=} \lim_{n\to\infty}\frac{\frac{1}{\ln^2(2)\; n\log(n)}}{\frac{1}{2\ln 2\; n\sqrt{\log n}}} \\
&= \lim_{n\to\infty}\frac{\sqrt{\log n}}{2\ln 2\log n} \\
&= \lim_{n\to\infty}\frac{1}{2\ln 2\sqrt{\log n}} \\
&= \frac{1}{2\ln 2\sqrt{\log\infty}} \\
&= \frac{1}{2\ln 2\sqrt{\infty}} \\
&= \frac{1}{\infty} \\
&= 0
\end{aligned}
$$

$\therefore\ g(n)=\sqrt{\log n}$ grows faster because $\log\log n = o(\sqrt{\log n})$

(b) Let's compare both functios as $n\to\infty$:

$$\lim_{n\to\infty}\frac{n!}{(n+1)!} = \lim_{n\to\infty}\frac{n!}{n!(n+1)} = \lim_{n\to\infty}\frac{1}{n+1} = \frac{1}{\infty} = 0$$

$\therefore (n+1)!$ grows faster because $n! = o(g((n+1)!))$.

(c) Let's compare both functions:

$$\lim_{n\to\infty}\frac{\frac{n}{\log n}}{\sqrt{n}} = \lim_{n\to\infty}\frac{n}{\sqrt{n}\log n} = \lim_{n\to\infty}\frac{\sqrt{n}}{\log n} \overset{\text{L'H}}{=} \lim_{n\to\infty}\frac{\frac{1}{2\sqrt{n}}}{\frac{1}{\ln 2n}} = \lim_{n\to\infty}\frac{ln\ 2\sqrt{n}}{2} = \infty$$

$\therefore \frac{n}{\log n}$ grows faster because $\frac{n}{\log n} = \omega(\sqrt{n})$

(d) Let's compare both functions as $n\to\infty$:

$$\lim_{n\to\infty}\frac{\log\log n}{\ln\ln n} \overset{\text{L'H}}{=} \lim_{n\to\infty}\frac{\frac{1}{\ln\ 2\log n}}{\frac{1}{n\ln n}} = \lim_{n\to\infty}\frac{\ln n}{\ln 2\log n} \overset{\text{L'H}}{=} \lim_{n\to\infty}\frac{1n}{\frac{\ln^2 n}{n}} = \lim_{n\to\infty}\frac{1}{\ln^2 2} \approx 2.081$$

$\therefore \log\log n = \Theta(\ln\ln n)$

5. $g_{10} \prec g_9 \sim g_{12} \prec g_{14} \prec g_{15} \sim g_6 \prec g_5 \prec g_{11} \prec g_{20} \prec g_{16} \prec g_{18} \sim g_7 \prec g_3 \sim g_{19} \prec g_4 \prec g_8 \sim g_{17} \prec g_1 \prec g_2 \prec g_{13}$

B Solve Recurrences

1. To apply Master Theorem, $T(n)$ must be defined by the recurrence $T(n) = aT(n/b) + f(n)$, for $a \geq 1$ and $b > 1$. But $b = 1$ for $T(n)$, then Master Theorem is not applicable. Because the base case of $T(n)$ is when $T(k) = c$, where k and c are constants. Then, when $n = 1$ we obtain:

$$\begin{aligned}
T(1) &= T(0) + 1 = c + 1 \\
T(2) &= T(1) + 4 = c + 1 + 4 \\
T(3) &= T(2) + 9 = c + 1 + 4 + 9 \\
T(4) &= T(3) + 16 = c + 1 + 4 + 9 + 16 \\
T(5) &= T(4) + 25 = c + 1 + 4 + 9 + 16 + 25 \\
&\vdots \\
T(n) &= T(n-1) + n^2 = c + 1 + 4 + 9 + 16 + 25 + \cdots + n^2
\end{aligned}$$

   This implies that $T(n) = c + \sum_{i=1}^{n} i^2 = c + \frac{n(n+1)(2n+1)}{6} = \frac{n^3}{3} + \frac{n^2}{2} + \frac{n}{6} + c = \Theta(n^3)$

$$\therefore T(n) = \Theta(n^3)$$

2. We have $a = 3$, $b = 2$, and $f(n) = n^2$, and thus we have that $n^{\log_b a} = n^{\log_2 3}$. Since $f(n) = \Omega(n^{\log_2 3 + \epsilon})$ for some $\epsilon > 0$, then case 3 applies if we can show that the regularity condition holds for $f(n)$. For suffitiently large n, we have that $af(n/b) = 3(n/2)^2 = (3/4)n^2 \leq (3/4)f(n)$ for $c = 3/4$.

$$\therefore T(n) = \Theta(n^2)$$

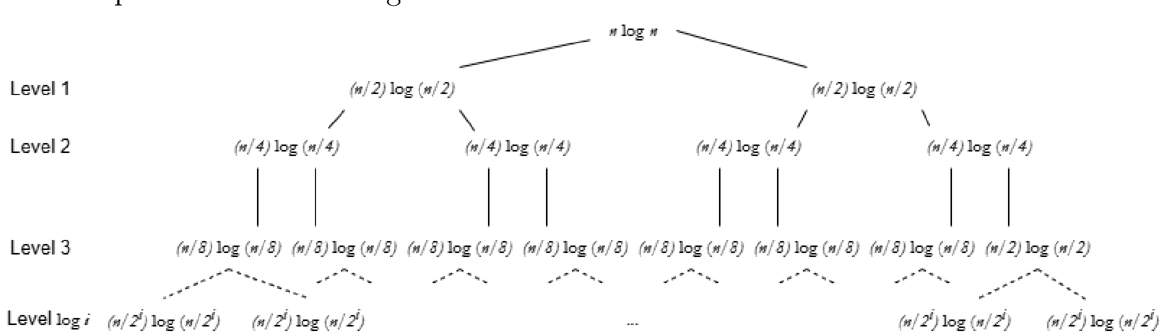
3. We have $a = 2$, $b = 4$, and $f(n) = \log\log n$, and thus we have that $n^{\log_b a} = n^{\log_4 2} = n^{\frac{\log_2 2}{\log_2 4}} = \sqrt{n}$. Since $f(n) = O(n^{\frac{1}{2} - \epsilon})$ for some $\epsilon > 0$, then case 1 applies.

$$\therefore T(n) = \Theta(\sqrt{n})$$

4. We have $a = 2$, $b = 2$, and $f(n) = n\log n$, and thus we have that $n^{\log_b a} = n^{\log_2 2} = n$. Since $f(n) \approx \Theta(n)$ differing by a factor of $\log^k n$ with $k = 1$, then case 2 applies.

$$\therefore T(n) = \Theta(n\log^2 n)$$

   Now we prove the same using a recursion tree:



The recursive tree depth is $log(n)$. If we follow each i-th recursion level for $i = 1, 2, \cdots, \log n$, we can see the pattern:

Level 1: $T(\frac{n}{2}) = 2(\frac{n}{2}\log\frac{n}{2}) = n\log n - n$
Level 2: $T(\frac{n}{4}) = 4(\frac{n}{2^2}\log\frac{n}{2^2}) = n\log\frac{n}{4} = n\log n - 2n$
Level 3: $T(\frac{n}{8}) = 8(\frac{n}{2^3}\log\frac{n}{2^3}) = n\log\frac{n}{8} = n\log n - 3n$

$\vdots$

Level $\log i$: $T(\frac{n}{2^i}) = 2^i(\frac{n}{2^i}\log\frac{n}{2^i}) = n\log\frac{n}{2^i} = n\log n - in$

Thus, the recursive function $T(n)$ is defined as:

$$
\begin{aligned}
T(n) &= \sum_{i=1}^{\log n}(n\log n - in) \\
&= n\log n(\log n + 1) - n\left(\frac{\log n(\log n+1)}{2}\right) \\
&= n\left(\log^2 n + \log n - \frac{\log^2 n}{2} - \frac{\log n}{2}\right) \\
&= n\left(\frac{log^2 n}{2} + \frac{\log n}{2}\right) \\
&= \frac{n}{2}(\log^2 n + \log n) \\
&= \Theta(n)\cdot\Theta(\log^2 n) \\
&= \Theta(n\log^2 n)
\end{aligned}
$$

$$\therefore T(n) = \Theta(n\log^2 n)$$

C Test the Candies

1. Pseudocode to discard bad candies

Algorithm 1: DiscardBadCandies(*candyBatch*[*n*], *n*)

```
Input: candyBatch with n candies and unknown c bad candies, where
       0 ≤ c ≤ n
Output: candyBatch without bad candies
if UseDevice(candyBatch) Returns "GOOD!" then
 |  return candyBatch
end
else if n > 1 then
 |  middle ← n/2
 |  left ← candyBatch[1 to middle]
 |  right ← candyBatch[middle + 1 to n]
 |  if UseDevice(left) Returns "BAD!" then
 |   |  left ← DiscardBadCandies(left, n/2)
 |  end
 |  if UseDevice(right) Returns "BAD!" then
 |   |  right ← DiscardBadCandies(right, n/2)
 |  end
 |  candyBatch ← (left) ∪ (right)
end
else
 |  remove candyBatch[1]
end
return candyBatch
```

2. When we use the device to test a batch of n candies and returns "BAD!", that means there are unknown c bad candies in the batch, assuming c is a constant where $0 \leq c \leq n$. Then we divide the batch into two sub-batches, S_1 and S_2, of size $n/2$. As we obtain these two sub-batches, one or both sub-batches contain at least one bad candy. The goal of my algorithm is to keep spliting in half the sub-batches at each recursion that contain at least one bad candy. In other words, at any level of the recursive tree, only sub-batches that contain at least one bad candy get splitted in half. Since the total number of bad candies is c at each recursion level, the number of sub-batches with at least one bad candyis at most c (since we can't split a sub-batch that passes as "GOOD!" by the device). When we obtain two sub-batches of size 1 by splitting a batch of size 2 for testing "BAD!" in the device, so either one candy is bad or both are. At that recursive level, the number of levels is $\log n$ because each sub-batch has been splited in halves, and since there are at most c bad candies, the number of tests has been made is $c \cdot \log n + 1$ (including the initial batch's test). Therefore, we need to do at least $c \log n = O(\log n)$ tests. ■

3. Because all candies are bad, each sub-batch S_i of candies will test with band candies and be splited in half obtaining 2^{i+1} new sub-batches at i-th recursion level for $i = 1, 2, \cdots, \log n$. However, the number of bad candies is unknown, so the algorithm will

keep spliting sub-batches in half until reach $\log n$-th recursion levels and obtaining n sub-batches of size 1. Thus, all candies has been tested by the device and identified as bad candies. The number of tests is given by:

$$\sum_{i=0}^{\log n} 2^i = 1 + 2 + 4 + 8 + \cdots + n = \frac{1 - 2^{\log(n)+1}}{1 - 2} = 2n - 1 = O(n) \qquad \blacksquare$$