Kaleb Kim
9/21/24

CSDS 310 Assignment 2

Note: Arrays are zero-indexed.

Problem 1

a) $\log^k n = o(n^\varepsilon)$, proven by the Limit Asymptotic Theorem:

$$\begin{aligned}\lim_{n\to\infty} \frac{f(n)}{g(n)} = \lim_{n\to\infty} \frac{\log^k n}{n^\varepsilon} &= \lim_{n\to\infty} \left(\frac{\log n}{n^{\varepsilon/k}}\right)^k \text{, applying L`Hopital`}s: \\ &\stackrel{\star}{=} \lim_{n\to\infty} \left(\frac{1/n}{\varepsilon n^{\varepsilon/k-1}}\right)^k \\ &= \left(\frac{0}{\varepsilon(\infty)^{\varepsilon/k-1}}\right)^k \\ &= 0\end{aligned}$$

This means that $g(n)$ is the asymptotic upper bound of $f(n)$. As this result does not depend on constants k, ε, this is a strict bound—which is also part of the Limit Asymptotic Theorem.

b) $n^k = o(c^n)$, proven by the Limit Asymptotic Theorem again:

$$\begin{aligned}\lim_{n\to\infty} \frac{f(n)}{g(n)} &= \lim_{n\to\infty} \frac{n^k}{c^n} \\ &\stackrel{\star}{=} \lim_{n\to\infty} \frac{kn^{k-1}}{c^n} \\ &\stackrel{\star}{=} \lim_{n\to\infty} \frac{k(k-1)n^{k-2}}{c^n}\end{aligned}$$

As the denominator remains fixed, this pattern will continue until:

$$\begin{aligned}\lim_{n\to\infty} \frac{k(k-1)(...)(2)(1)n^1}{c^n} &\stackrel{\star}{=} \lim_{n\to\infty} \frac{k(k-1)(...)(2)}{c^n} \\ &= 0\end{aligned}$$

By the same reason as part a), this result proves that $g(n)$ is the asymptotic strict upper bound of $f(n)$.

c) There is no asymptotic relation between the two. If we were to prove this through the closest definition of Big-O notation, stating there exists some constant $a > 0$ that satisfies the following:

$$\begin{gathered}0 \le f(n) \le ag(n) \\ 0 \le \left(\sqrt{n}\right) \le a\left(n^{\sin(n)}\right) \\ 0 \le n^{\frac{1}{2}} \le an^{\sin(n)} \\ 0 \le \ln\left(n^{\frac{1}{2}}\right) \le \ln\left(an^{\sin(n)}\right) \\ 0 \le \frac{1}{2} \le a\sin(n)\end{gathered}$$

Regardless of what we choose for a, $\sin(n)$ can be 0 for sufficiently large inputs for n, making the inequality false. If we were to try other notation definitions, there will still be no bound.

Problem 2

a) True. By the definition of Big O, with $a \in \mathbb{R}$ such that $a > 0$:

$$\begin{gathered} 0 \leq f(n) \leq ag(n) \\ 0 \leq f(n) + c \leq af(n) \\ 0 \leq 1 \leq \frac{af(n)}{f(n) + c} \end{gathered}$$

Using a limit to evaluate a sufficiently large n:

$$\begin{aligned} \lim_{n \to \infty} \frac{af(n)}{f(n) + c} &\overset{\star}{=} \lim_{n \to \infty} \frac{af'(n)}{f'(n)} \\ &= \lim_{n \to \infty} a \\ &= a \end{aligned}$$

Since there exists a such that $a \geq 1$ to make the inequality will hold true, this proves the original statement.

b) False. For $f(n) = 2^n$, by the definition of Theta Notation $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, with c_1, c_2 are positive and n is sufficiently large. We have $g(n) = f(2n) = 2^{2n} = (2^n)^2$. Substituting:

$$\begin{gathered} 0 \leq c_1 (2^n)^2 \leq 2^n \leq c_2 (2^n)^2 \\ 0 \leq c_1 2^n \leq 1 \leq c_2 2^n \end{gathered}$$

No value of c_1, c_2 can fix this inequality the way I can fix her for any sufficiently large n. Thus, the statement is false.

c) True. Let $g(n) = f(2n) = (2n)^c = 2^c n^c$. By the definition of Big O, with $a \in \mathbb{R}$ such that $a > 0$:

$$\begin{gathered} 0 \leq f(n) \leq ag(n) \\ 0 \leq n^c \leq a(2^c n^c) \\ 0 \leq 1 \leq a \cdot 2^c \end{gathered}$$

Suppose $a = 1$ and given that $c > 0$, for any sufficiently large value of n the inequality holds true. Thus, there exists a constant which holds the statement true.

Problem 3

a) Converting $T(n)$ to Master Theorem form, $T(n) = aT\left(\frac{n}{b}\right) + \Theta\left(n^d\right)$. This falls under Case 2, thus $T(n) = \Theta(n \log n)$

b) yeah

Problem 4

This is an incomplete merge sort. So let's complete it. We'll use two procedures (methods) to accomplish this: one to loop through the entire array until A contains one subarray of length n.

procedure MERGE(A, n):
  if n_A = 1:
    return A
  A' $\leftarrow$ Array of $\left\lceil \frac{n}{2} \right\rceil$ elements

```
    for 0 ≤ i ≤ n − 1:
        A'[i] ⟵ SUBMERGE(A[2i], A[2i+1])
    return MERGE(A')
```

Then, the merging of two subarrays, cleverly named:

```
procedure SUBMERGE(A, B):
    i ⟵ 0
    j ⟵ 0
    C ⟵ Array of n_A + n_B elements
    while i + j < len(A) + len(B):
        if i == len(A):
            C[i+j] ⟵ B[j]
            j ⟵ j + 1
        else if j == len(B) or A[i] < B[j]:
            C[i+j] ⟵ A[i]
            i ⟵ i + 1
        else:
            C[i+j] ⟵ B[j]
            j ⟵ j + 1
    endwhile
    return C
```

Does this need a proof? Yeah, probably