CS 413 Homework 2

Sterling Jeppson

February 6, 2021

Problem 1

a. n^2

  i. $\dfrac{(2n)^2}{n^2} = \dfrac{4n^2}{n^2} = 4$ times as long.

  ii. $\dfrac{(n+1)^2}{n^2} = \dfrac{n^2+2n+1}{n^2} = 1 + \dfrac{2}{n} + \dfrac{1}{n^2}$ times as long.

b. n^3

  i. $\dfrac{(2n)^3}{n^3} = \dfrac{8n^3}{n^3} = 8$ times as long.

  ii. $\dfrac{(n+1)^3}{n^3} = \dfrac{n^3+3n^2+3n+1}{n^3} = 1 + \dfrac{3}{n} + \dfrac{3}{n^2} + \dfrac{1}{n^3}$ times as long.

c. $100n^2$

  i. Since 100 is a constant factor it can be cancelled from the numerator and denominator. Hence, by part a, the run time will be 4 times as long.

  ii. By part a, the run time will be $1 + \dfrac{2}{n} + \dfrac{1}{n^2}$ times as long.

d. $n \log n$

  i.
$$\begin{aligned}
\frac{2n \log 2n}{n \log n} &= \frac{2 \log 2n}{\log n} \\
&= 2 \log_n 2n \\
&= 2(\log_n 2 + \log_n n) \\
&= 2(\log_n 2 + 1) \\
&= 2 \log_n 2 + 2 \\
&= \log_n 2^2 + 2 \\
&= \log_n 4 + 2 \text{ times as long.}
\end{aligned}$$

  ii. $\dfrac{(n+1)\log(n+1)}{n \log n} = \dfrac{n+1}{n} \log_n (n+1)$ times as long.

e. 2^n

   i. $\dfrac{2^{2n}}{2^n} = 2^{2n-n} = 2^n$ times as long.

   ii. $\dfrac{2^{n+1}}{2^n} = 2^{n+1-n} = 2$ times as long.

Problem 2

Since we have at most one hour to compute a result we need to know the number of operations that the computer can execute in one hour.

$$\frac{10^{10}\text{ operations}}{1\text{ second}} \cdot \frac{60\text{ seconds}}{1\text{ minute}} \cdot \frac{60\text{ minutes}}{1\text{ hour}} = \frac{60^2 \cdot 10^{10}\text{ operations}}{1\text{ hour}}$$

Now we solve the functions a-f below for n.

a. $n^2 = 60^2 \cdot 10^{10} = 60^2 \cdot (10^5)^2 = (60 \cdot 10^5)^2 \implies n = 60 \cdot 10^5$.

b. $n^3 = 60^2 \cdot 10^{10} \implies n \approx 33019.3$. Since we have a max of 1 hour we must round down our result to the nearest integer to obtain $n = 33019$.

c. $100n^2 = 10^2 n^2 = (10n)^2 = 60^2 \cdot 10^{10}$. Now by part a, $10n = 60 \cdot 10^5 \implies n = 60 \cdot 10^4$.

d. $n \log_2 n = 60^2 \cdot 10^{10}$. To form our approximate solution we will use Newton's Method. First form the equation

$$f(x) = x \cdot \log_2 x - 60^2 \cdot 10^{10}$$

Guess an approximate root, x_0 and form the linearization of $f(x)$ at x_0. We do this because the tangent line of a differentiable function at x_0 is a close approximation to the function near the point $(x_0, f(x_0))$. Then if we have chosen a good initial approximate root, x_0, the root of the tangent line, x_1, will be closer to the true root of the function.

$$f(x_1) - f(x_0) = f'(x_0)(x_1 - x_0)$$

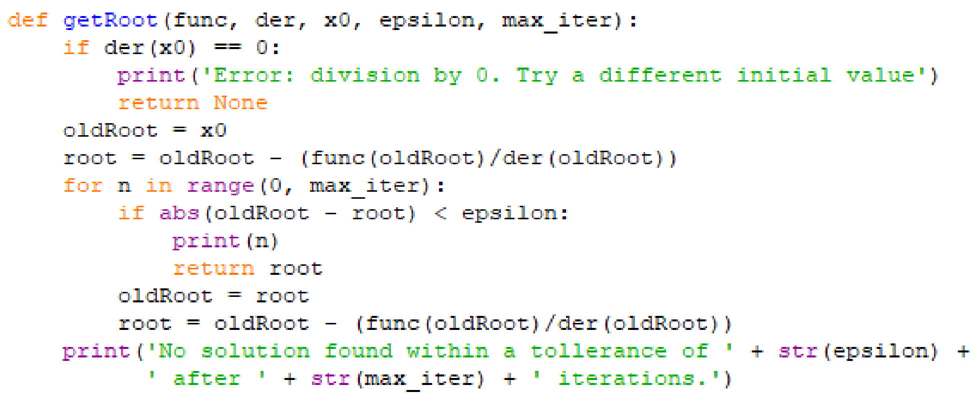
We want to find the root of this line so let $f(x_1) = 0$ and solve for x_1 to obtain

$$\textit{newRoot} = x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}$$

Now repeat this process to obtain closer and closer approximations of the root.

$$\textit{newRoot} = x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Stop once $|x_{n+1} - x_n| < \epsilon$ a user defined tolerance after which not much benefit is deemed to be gained by repeating the formula. This translates nicely to a simple python function implemented below.



```python
def getRoot(func, der, x0, epsilon, max_iter):
    if der(x0) == 0:
        print('Error: division by 0. Try a different initial value')
        return None
    oldRoot = x0
    root = oldRoot - (func(oldRoot)/der(oldRoot))
    for n in range(0, max_iter):
        if abs(oldRoot - root) < epsilon:
            print(n)
            return root
        oldRoot = root
        root = oldRoot - (func(oldRoot)/der(oldRoot))
    print('No solution found within a tollerance of ' + str(epsilon) +
          ' after ' + str(max_iter) + ' iterations.')
```

Figure 1: Newtons Method Python Function

Now we need to take the derivative of $f(x)$. By simple calculus

$$\begin{aligned}\frac{d}{dx}[x\log_2 x - 60^2\cdot 10^{10}] &= \frac{d}{dx}[x\log_2 x] - \frac{d}{dx}[60^2\cdot 10^{10}]\\ &= \frac{d}{dx}[x]\cdot\log_2 x + x\cdot\frac{d}{dx}[\log_2 x] - 0\\ &= 1\cdot\log_2 x + x\cdot\frac{1}{x\ln 2}\\ &= \log_2 x + \frac{1}{\ln 2}\\ &= \frac{\ln x}{\ln 2} + \frac{1}{\ln 2}\\ &= \frac{\ln x + 1}{\ln 2}\end{aligned}$$

The last item that we need is a good approximation for the root of the equation $f(x) = x\cdot\log_2 x - 60^2\cdot 10^{10}$. Observe that exact solutions to equations of the form $f(x) = x\cdot\log_2 x - i\cdot 2^i$ are of the form 2^i where i is an integer. Since $39\cdot 2^{39} < 60^2\cdot 10^{10} < 40\cdot 2^{40}$, we can conclude that $2^{39} < x < 2^{40}$. Now for the python function let $x_0 = 2^{39}$, $\epsilon = .001$, and max_iter $= 10$. The algorithm produces the result *root* $= 906316482853.1769$. Since n must be an integer round down to $n = 906316482853$. Plugging n back into the expression $n\log_2 n$ gives $35999999999992.71 < 60^2\cdot 10^{10}$. Plugging $n+1$ into the expression $n\log_2 n$ gives $36000000000033.875 > 60^2\cdot 10^{10}$. Hence the algorithm worked and we can conclude that $n = 906316482853$.

e. $2^n = 60^2\cdot 10^{10} \implies \log_2 2^n = \log_2 60^2\cdot 10^{10} \implies n \approx 45.03$. Since n must be an integer we will say $n = 45$.

f. $2^{2^n} = 60^2\cdot 10^{10}$. Substitute n for $2n$. Then by part e, $2^n = 45$. Take $\log_2$ of both sides to obtain $n = \log_2 45 \approx 5.49$. Since n must be an integer we will say $n = 5$.

Problem 3

Since f_1, f_2, and f_3 are polynomial time functions, by 2.7 we can order them by their highest degree. Since the degrees are $2.5, .5$, and 1 respectively, we have that f_2 is $O(f_3)$ and f_3 is $O(f_1)$. We need to determine the placement of f_6 relative to f_1. Since $f_1(n) = n^{2.5} = n^2 \cdot n^{1/2}$ and $f_6(n) = n^2 \log n$, the question to ask is whether $n^{1/2}$ grows faster than $\log n$. But by 2.8, for every $b > 1$ and every $x > 0$, $\log_b n = O(n^x)$. Hence $n^{1/2}$ grows faster than $\log n$ and so f_6 goes right before f_1. By 2.9, every exponential grows faster than every polynomial and so f_4 and f_5 will be the functions with the fastest growth rates. Since there exists an n_0, for example 1, such that for all $n > n_0$, $1 \leq 10^n$, it follows that $10^n \leq 10^n \cdot 10^n = 100^n$. Hence f_4 is $O(f_5)$. Our finale ordering from slowest rate of growth to quickest is f_2, f_3, f_6, f_1, f_4, and f_5.

Problem 4

g_1 is $O(g_3)$. Suppose that this is so for a constant multiplier of 1 for all $n > n_0$. Then,

$$\begin{aligned}
2^{\sqrt{\log n}} &< n(\log n)^3 && \text{by supposition}\\
\sqrt{\log n} &< \log_2\left(n(\log n)^3\right) && \text{take } \log_2 \text{ of both sides}\\
\sqrt{\log n} &< \frac{\log\left(n(\log n)^3\right)}{\log 2} && \log_c x = \frac{\log x}{\log c}\\
\sqrt{\log n} &< \frac{\log n + \log\left((\log n)^3\right)}{\log 2} && \log(xy) = \log x + \log y\\
1 &< \frac{\log n + \log\left((\log n)^3\right)}{\sqrt{\log n}\cdot \log 2} && \text{divide both sides by } \sqrt{\log n}\\
1 &< \frac{\log n}{\sqrt{\log n}\cdot \log 2} + \frac{\log\left((\log n)^3\right)}{\sqrt{\log n}\cdot \log 2}\\
1 &< \frac{\sqrt{\log n}}{\log 2} + \frac{\log\left((\log n)^3\right)}{\sqrt{\log n}\cdot \log 2}
\end{aligned}$$

Now the left fraction on the right side of the inequality goes to infinity. The right fraction on the right side of the inequality does not go to negative infinity when n is large. It must then go to a constant or positive infinity. In either case the expression is larger than 1 at some point. Strictly speaking this is an instance of the converse error. Note that $P \implies Q \not\equiv Q \implies P$. However, in this case, you can trace the last true statement derived back up to the conclusion. I just don't want to rewrite anything. Next we have g_3 is $O(g_4)$. To see why this is true note that $g_4 = n^{4/3} = n \cdot n^{1/3}$ and $g_3 = n(\log n)^3$. The n multiplier will provide the same rate of growth to each function so we only need to know if $n^{1/3}$ or $(\log n)^3$ grows faster. By taking the cubed root of both sides we are now comparing $n^{1/9}$ and $\log n$. But by 2.8, $\log n$ is $O(n^x)$ and so g_3 is $O(g_4)$. Next g_4 is $O(g_5)$. This is because they are both n raised to a power but g_5 is raised to $\log n$ which goes to infinity while g_4 is raised to a constant. Next g_5 is

$O(g_2)$. Again we will commit a logical fallacy to see why this is true.

$$\begin{aligned}
n^{\log n} &< 2^n && \text{by supposition} \\
\log n &< n \cdot \log_n 2 && \text{take } \log_n \text{ of both sides} \\
\log n &< n \cdot \frac{\log 2}{\log n} && \log_c x = \frac{\log x}{\log c} \\
\frac{(\log n)^2}{\log 2} &< n \\
\frac{\log n}{\sqrt{\log 2}} &< \sqrt{n}
\end{aligned}$$

But this must be true for all n greater than some n_0 by 2.8. Now trace from the true statement back to the conclusion and it follows that g_5 is $O(g_2)$. Next we have that g_2 is $O(g_7)$ and g_7 is $O(g_6)$. This is easy to see. They all have the same base, namely 2. But they are raised to different powers. We need to compare the powers which are n, n^2, and 2^n. By theorem on polynomial orders n^2 grows faster than n and by 2.9 every exponential grows faster than every polynomial. Finally we list the results from slowest to fastest rate of growth: $g_1, g_3, g_4, g_5, g_2, g_7, g_6$.