Samuel Kalisch sk54596
CS313E
Assignment 4

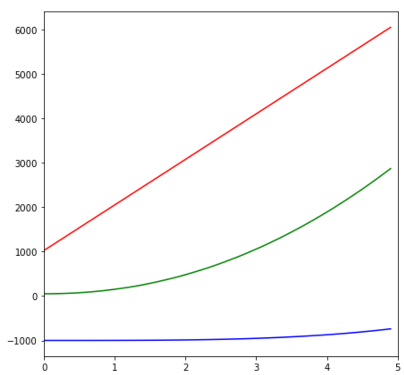
Tasks

1. Use python to create 3 different plots of the following functions (15 points):

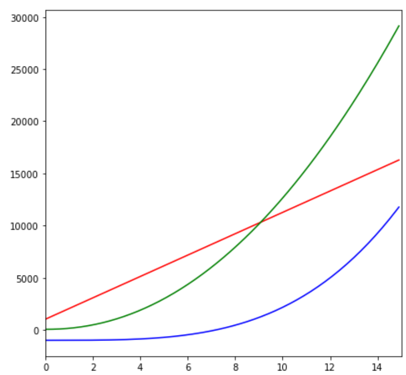
$$f_1(n) = (2^{10})(n) + 2^{10}$$
$$f_2(n) = n^{(3.5)} - 1000$$
$$f_3(n) = 100n^{(2.1)} + 50$$

Vizualiation:

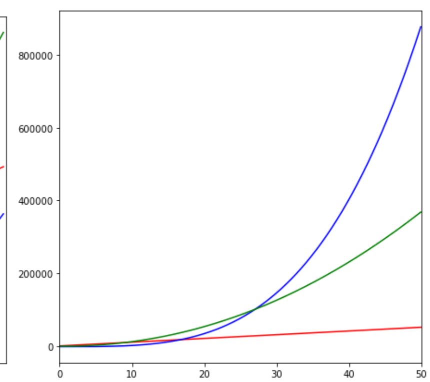
1.



2.



3.



Description:

1. In the first graph, F1 (red) is above the other two, F3 (green) is between F2 (blue) and F1(red), and finally F2 (blue) is at the bottom. We can see than red is increasing at a constant rate and that green and blue are not. Green has the highest increasing rate approaching to 5.
2. In the second graph, F3 (green) is above the other two, F1 (red) is between F2 (blue) and F3 (green), and finally F2 (blue) is still at the bottom. We can see than still, red is increasing at a constant rate and that green and blue are not. Green and blue have similar increasing rates approaching to 15.
3. Finally, F2 (blue) is more than double than any other function and is clearly at the top, F3 (green) is between F2 (blue) and F1(red), and finally F1 (red) is below the other two. We can see than still, red is increasing at a constant rate and that green and blue are not. Blue has skyrocketed and has the highest increasing rate approaching to 50.

Code:

```
import math
import numpy as np
import matplotlib.pyplot as plt

for msize in [5, 15, 50]:
  n = np.arange(0, msize, 0.1)

  plt.plot(n, (2**10)*n + 2**10 , 'red', n, n**3.5 - 1000, 'blue', n, 100*n**2.1 + 50, 'green')

  plt.xlim(0, msize)
  plt.rcParams["figure.figsize"] = (7,7)
  plt.show()
```

2. Asymptotic Notation. (15 points)

Describe your answer.

- Is $2^{(n+1.3)} = O(2^n)$?
- Is $3^{(2\times n)} = O(3^n)$?

1. **TRUE**. If lim n->∞(f(x)/g(x)) ≠ ∞, then f(x) = O(g(x)). In this case 2^(n+1.3) = O(2^n).

$$\lim_{n\to\infty}\left(\frac{2^{(n+1.3)}}{2^n}\right) = \lim_{n\to\infty}(2^{1.3}) = 2^{1.3}$$

2. **False**. If lim n->∞(f(x)/g(x)) ≠ ∞ , then f(x) = O(g(x)). In this case 3^(2n) = O(3n).

$$\lim_{n\to\infty}\left(\frac{3^{(2n)}}{3^n}\right) = \lim_{n\to\infty}\left(\frac{9^n}{3^n}\right) = \infty$$

3. For each pair of functions f (n) and g(n), check if f (n) = O(g(n))?

Functions f (n) and g(n) are:

1. $f(n) = (4\times n)^{150} + (2\times n + 1024)^{400}$ vs. $g(n) = 20\times n^{400} + (n+1024)^{200}$
2. $f(n) = n^{1.4}\times 4^n$ vs. $g(n) = n^{200}\times 3.99^n$
3. $f(n) = 2^{log(n)}$ vs. $g(n) = n^{1024}$

1. TRUE. If lim n->∞(f(x)/g(x)) ≠ ∞ , then f(x) = O(g(x)).

$$\lim_{n\to\infty}\left(\frac{4n^{150}+(2n+1024)^{400}}{20n^{400}+(n+1024)^{200}}\right)=\lim_{n\to\infty}\left(\frac{(2n+1024)^{400}}{20n^{400}}\right)=\lim_{n\to\infty}\left(\frac{(2n+1024)^{400}}{20n^{400}}\right)$$

$$=\lim_{n\to\infty}\left(\frac{2^{400}n^{400}}{20n^{400}}\right)=\lim_{n\to\infty}\left(\frac{2^{400}}{20}\right)=\frac{2^{400}}{20}$$

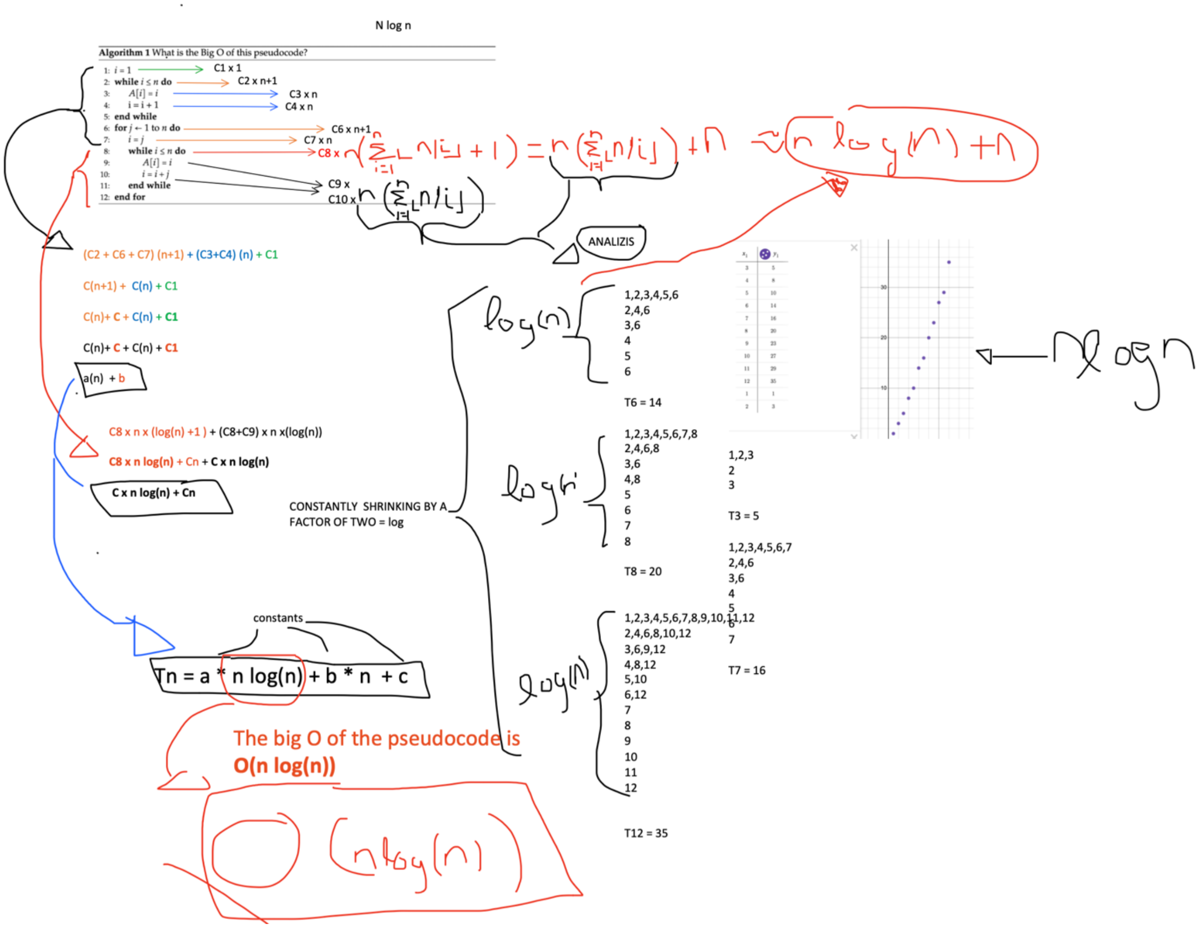
2. FALSE. If lim n->∞(f(x)/g(x)) ≠ ∞ , then f(x) = O(g(x)).

$$\lim_{n\to\infty}\left(\frac{n^{1.4}4^{n}}{n^{200}3.99^{n}}\right)=\lim_{n\to\infty}\left(\frac{4^{n}}{3.99^{n}}\right)=\infty$$

3. TRUE. If lim n->∞(f(x)/g(x)) ≠ ∞ , then f(x) = O(g(x)).

$$\lim_{n\to\infty}\left(\frac{2^{\log(n)}}{n^{1024}}\right)=0$$

4. Analyze the Algorithm 1 and give a Big O bound on the running time as a function of n. Carefully describe your justifications. (20 points)



To get the Big O of this pseudocode I examined the code, specifically the while loop inside the for loop. This nested loop specifically is where the biggest cost of the program is. The first loop runs n times which means the second one runs n times the cost of it. After an analysis of the second loop I concluded it ran a logarithmic pattern because of the way the numbers shrank consistently, this meant the Big O was **O(n * log (n)).**

5. Analyze the Algorithm 2. What is the Big O on the running time as a function of n. Carefully describe your justifications. (20 points)

ALGO 2

Monday, February 21, 2022 3:14 PM

Algorithm 2 What is the Big O of this pseudocode?

```
x = 0                          → C1 x 1
for i ← 0 to n do              → C2 x n
    for j ← 0 to (i × n) do    → C3 x n x
        x = x + 10             → C4 x (n) x
    end for
end for
```

C3 x n x $\sum_{i=0}^{n} i = C3 \times n \times \left(\frac{n(n+1)}{2}\right)$

C4 x (n) x $\sum_{i=0}^{n} (i-1) = C4 \times n \times \left(\frac{n(n+1)}{2} - n\right)$

6x1 = 6
6x2 = 12
6x3 = 18
6x4 = 24
6x5 = 30
6x6 = 36

6 x 21 = 126

The big O of the pseudocode is **O(n **3)**

$\boxed{O(n\text{^}3)}$

C4 x n x ((n(n+1)/2) -n) + C3 x n x ((n(n+1)/2)) + (C2) (n) + C1x1

6*(6+1) = 42

C4 x n x ((n**2+n)/2) - n) + C3 x n x ((n**2+n)/2)) + (C2) (n) + C1

C4 x ((n**3+n**2)/2) - n**2) + C3 x ((n**3+n**2)/2)) + (C2) (n) + C

C4 x (n**3/2) + C4 x (n**2)/2 - C4 x n**2+ C3 x (n**3)/2 + C3 x (n**2)/2 + (C2) (n) + C1

C4 x (n**2)/2 - C4 x n**2 + C3 x (n**2)/2 = 0

C x 2 n**3 + (C2) (n) + C1x1

C(n**3)+ C(n) + C

a(n**3)+ b(n) + c

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2},$$

To get the Big O of this pseudocode I examined the code, specifically the for loop. After an analysis I concluded it ran a cubic pattern this meant the Big O was **O(n **3).**