CS313E - Software Design
Assignment 4 - 100 Points

Please note the honor code policy regarding this Assignment. You may not discuss particular questions or discuss or transmit answers from the assignment with other people, except for the CS313e teaching team.

Please submit to the Gradescope!

Note: You can create a single PDF file with multiple pages and submit it to the Gradescope. You can use using any software like MS-Word or *LaTeX* to include mathematic formulas if needed. If you can not create a PDF you can write on a piece of paper, scan it or make a picture of it using your smartphone and submit multiple images into Gradescope.

You can use this guide to see how you can submit your assignment.
https://gradescope-static-assets.s3.amazonaws.com/help/submitting_hw_guide.pdf

You can use this video tutorial to know how to submit your grade to Gradescope
https://www.youtube.com/watch?v=u-pK4GzpId0.

Tasks

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

$$f_1(n) = (2^{20}) * (n) + 2$$

$$f_2(n) = n^{(7.1)} + (2.1)^{20}$$

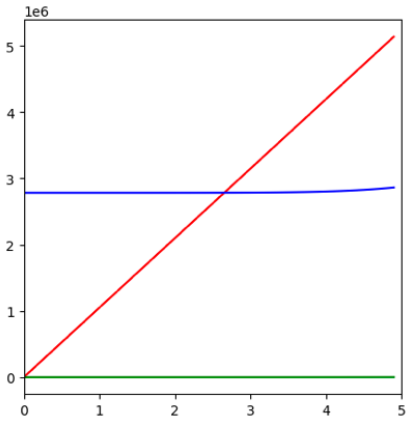
$$f_3(n) = 4^n - (2.1)^8$$

- Create 3 plots and limit the horizontal x-axis to $n = 5, 15, 50$. On each of the 3 plots you need to show the above 3 functions. On the first plot the x-axis is limited to 5, on second one x-axis is limited to 15 and on the 3rd one x-axis is limited to 50.
- Visualize the 3 functions in 3 colors (f_1 in red, f_2 in blue, f_3 in green).
- Describe your visualization and what you see in these 3 plots.
- Add your visualization and your python code to your PDF report file.

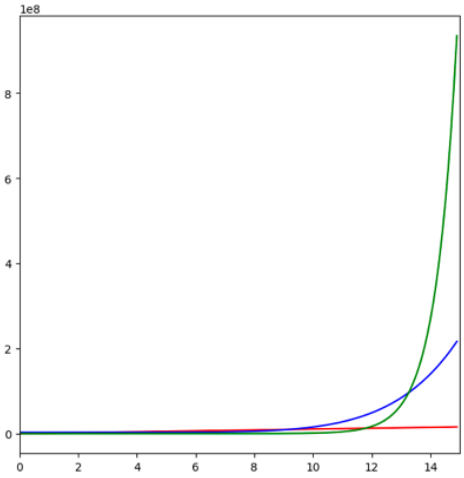
You can use the template implementation provided in class.

Here https://colab.research.google.com/drive/1Th23RUUaMKbM3CHab6RgVV2RFwTT1M_9?usp=sharing

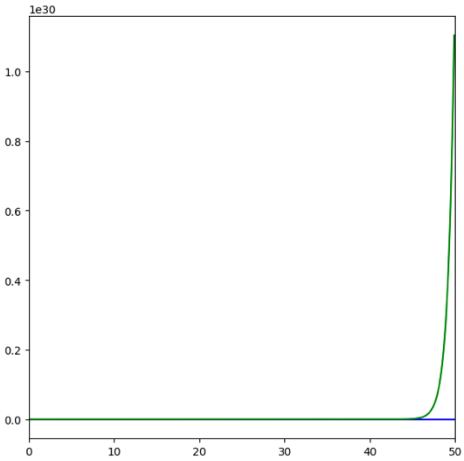
You can run this task on google Collaboratory.



This is the plot where x is limited to 5. In this plot we can see that within the constraints 0 < x < 5, $f_1(n)$ is the function that has the highest growth rate out of the 3 functions. The next highest growth rate function is $f_2(n)$ within the scope of this graph and $f_3(n)$ is the lowest growth rate function in the scope of this graph.



This is the plot where x is now limited to 15, and in this plot, we can see that $f_1(n)$ is no longer has the greatest growth rate and in fact $f_3(n)$ now has the highest growth rate by a large margin even compared to $f_2(n)$ which also has a growth rate that is greater than $f_1(n)$.



In this plot where x is limited to 50, we can barely see $f_1(n)$ and $f_2(n)$ since their respective growth rates are significantly lower than that of $f_3(n)$. This plot shows that at larger values of n, $f_3(n)$ outgrows the other two functions very rapidly.

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

size_1 = 5

n = np.arange(0, size_1, 0.1)
plt.plot(n, (2**20) * n + 2 , 'red', n, n**7.1 + 2.1**20, 'blue', n, 4**n - 2.1**8,
'green')
plt.xlim(0, size_1)
plt.rcParams["figure.figsize"] = (7,7)
plt.show()

size_2 = 15

n = np.arange(0, size_2, 0.1)
plt.plot(n, (2**20) * n + 2 , 'red', n, n**7.1 + 2.1**20, 'blue', n, 4**n - 2.1**8,
'green')
plt.xlim(0, size_2)
plt.rcParams["figure.figsize"] = (7,7)
```

```
plt.show()

size_3 = 50

n = np.arange(0, size_3, 0.1)
plt.plot(n, (2**20) * n + 2 , 'red', n, n**7.1 + 2.1**20, 'blue', n, 4**n - 2.1**8,
'green')
plt.xlim(0, size_3)
plt.rcParams["figure.figsize"] = (7,7)
plt.show()
```

---

2. Asymptotic Notation. **(15 points)**

- Is $2^{(2n+2.3)} = O(2^n)$?
- Is $3^{(2\times n)} = O(3^n)$?

Describe your answer.

$$2^{2n+2.3} = O(2^n)$$

Proof by definition of Big-O notation:

$$2^{2n+2.3} \leq c \times 2^n$$
$$2^{2n} \times 2^{2.3} \leq c \times 2^n$$
$$2^{2.3} \times 2^{2n} \leq c \times 2^n$$
$$2^{2.3} \times (2^n)^2 \leq c \times 2^n$$
$$2^{2.3} \times 2^n \leq c$$

$\therefore$ *There is no constant c such that* $0 \leq f(n) \leq c \cdot g(n)$ *for all* $n > n_0$, *so* $2^{2n+2.3} \neq O(2^n)$

$$3^{2\times n} = O(3^n)$$

Proof by definition of Big-O notation:

$$3^{2\times n} \leq c \times 3^n$$
$$(3^n)^2 \leq c \times 3^n$$
$$3^n \leq c$$

$\therefore$ *There is no constant c such that* $0 \leq f(n) \leq c \cdot g(n)$ *for all* $n > n_0$ *so* $3^{2\times n} \neq O(3^n)$

---

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. $g(n) = 20 \times n^{300} + (n + 121)^{152}$
2. $f(n) = n^{1.4} \times 4^{2n}$ vs. $g(n) = n^{100} \times 3.99^{n}$
3. $f(n) = 2^{log_2^n}$ vs. $g(n) = n^{1024}$

Describe your justifications. **(30 points)**

1) $f(n) = (4 \times n)^{150} + (2 \times n + 1024)^{400}$

$= 4n^{150} + (2n + 1024)^{400}$

$= 4n^{150} + (2n)^{400} + \ldots + 1024^{400}$

$f(n)$ highest term: $(2n)^{400}$

$g(n) = 20 \times n^{300} + (n+121)^{152}$

$= 20n^{300} + (n+121)^{152}$

$= 20n^{300} + n^{152} + \ldots + 121^{152}$

$g(n)$ highest term: $20n^{300}$

By Big-O limit definition:

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{(2n)^{400}}{20n^{300}} = \infty$$

$\therefore$ In this case, $f(n) \neq O(g(n))$

---

2) $f(n) = n^{1.4} \times 4^{2n}$

$= n^{1.4} \times 16^{n}$

$f(n)$ highest term: 16^n

$g(n) = n^{100} \times 3.99^{n}$

$g(n)$ highest term: n^{100}

$\therefore$ since $g(n) > f(n)$, $0 \leq f(n) \leq c \cdot g(n)$ for some constant $c, n_0 > 0$ for all $n > n_0$ and $f(n) = O(g(n))$

---

3) $f(n) = 2^{\log_2 n} = n$

$g(n) = n^{1024}$

By Big-O limit definition,

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = \frac{n}{n^{1024}} = \frac{1}{n^{1023}} = 0$$

$\therefore$ In this case, $f(n) = O(g(n))$ since $0 \leq f(n) \leq c \cdot g(n)$ for some constant $c, n_0 > 0$ for all $n > n_0$

---

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)**

Algorithm 1 What is the Big O of this pseudocode?

```
1: i = 1
2: while i ≤ n do
3:     A[i] = i
4:     i = i + 1
5: end while
6: for j ← 1 to n do
7:     i = j
8:     while i ≤ n do
9:         A[i] = i
10:        i = i + j
11:    end while
12: end for
```

Explanation:

The Big-O of this code is $O(n^2 \cdot log(n))$. This is because the first while loop runs n times, and the nested loop runs $n \times \sum_{j=1}^{n} \frac{n}{j}$ times which leads to a runtime cost of $T(n) = n + n \cdot \sum_{j=1}^{n} \frac{n}{j}$. Simplifying this function we then get $T(n) = n + n^2 log(n)$. Therefore since $n^2 log(n)$ is the largest term and contributes the most to the growth of the function, $O(n^2 log(n))$ best describes the runtime of this algorithm.

---

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)**

Algorithm 2 What is the Big O of this pseudocode?

```
1: x = 0
2: for i ← 0 to n do
3:     for j ← 0 to (i × n) do
4:         x = x + 10
5:     end for
6: end for
```

Explanation:

The Big-O of this code is $O(n^4)$. This is because the outer loop runs (n+1) times, and the inner loop runs $n \cdot \sum_{j=0}^{n} j$ which leads to a runtime cost of $T(n) = (n+1) \cdot n \sum_{j=0}^{n} j$. Simplifying this function, we then get T(n) = $\frac{1}{2}(n^4 + 2n^3 + n^2)$. Therefore since n^4 is the largest term and contributes the most to the growth to the function, $O(n^4)$ best describes the runtime of this algorithm