CS566 Assignment 1 Fall 2020

Documentation: **CS 566 Algorithms**

- Assignment 1
  Author: Phillip.Escandon@bu.edu
  Date: Sept 9, 2020

5 Questions

1. Plot 3 functions
2. Asymptotic Behavior
3. Pairs of functions
4. Analyze and give upper / lower bounds for f(x)
5. Analyze and describe big O for f(x)

Standard Python Imports

```
env: PYTHONBREAKPOINT=IPython.core.debugger.set_trace
```

Out[48]: Click here to toggle on/off the raw code.

Create 3 plots of the following functions

$$f(x) = 2^{10}(x) + 2^{10}$$
$$f(x) = x^{3.5} - 1000$$
$$f(x) = 100x^{2.1} + 50$$

with
x = 5
x = 15
x = 50

N = 5 Data

Create F(x) definitions

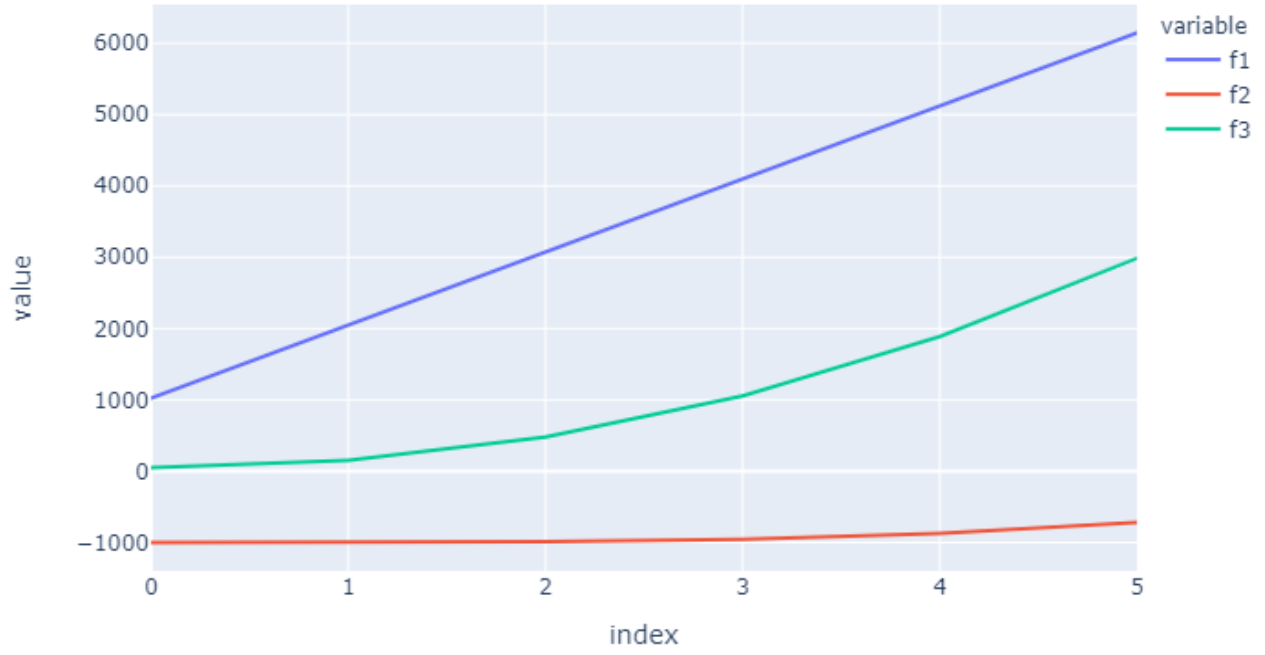
Create some dataframes with data from each function

Using the Pandas Dataframe.

Concatenate the data into one dataframe

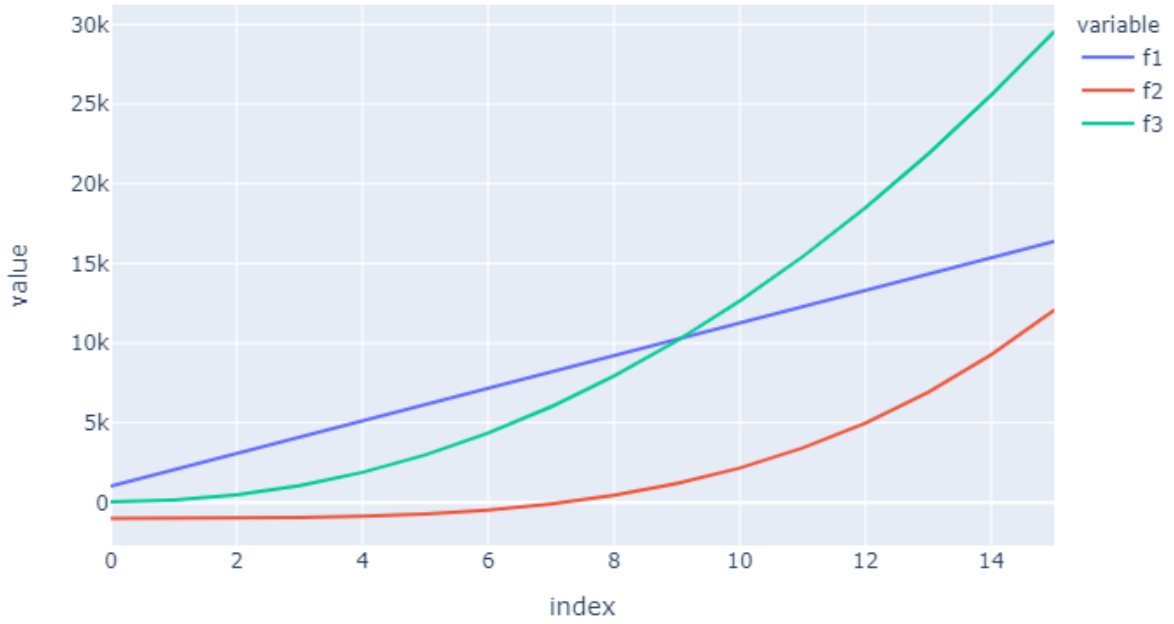
Using the concatenate function from pandas

N = 5 Plot



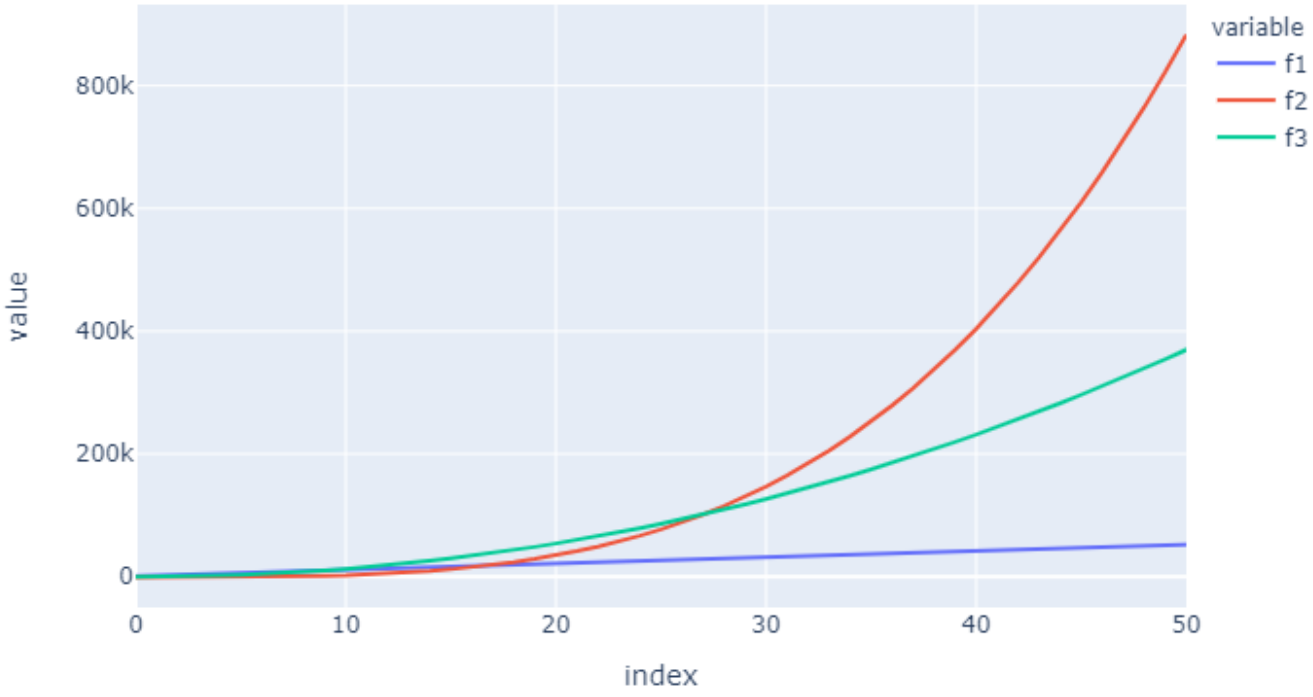
Description:
In this plot we are only looking at a 'small' data size, n = 5.
For small amounts of data, function 2 (red) is best, as its 'growth rate or slope is close to zero.
Function 1 (blue) also appears linear, which would give me confidence that it could possibly stay the same over larger data sets applied.
Function 3 (green) appears to begin growing as the data size increases. Big red flag if this is seen even in a small data size.

N = 15 Plot



Description:
With the data size increased to 15 we can begin to see the difference between the functions. F3 is definately growing faster than the other functions, and F1 appears linear. So for a data set size of 15, F2 would be the best.

N = 50 Plot



Description:
In this case, the data size is now 50 and we can now see that for large datasets, F1 performs the best with F2 and F3 growing at a quadratic (or greater) rate. Reviewing the functions, it is true that F1 is O(n), F2 is O(n^3.5) and F3 is O(n^2.1). So for larger datasets, F1 would have the best execution time.

Asymptotic Notation

- Is 2^{n+1.3} = O(2^n)
  Yes, the constants will fall away with the Big O notation and we will be left will O(2^n).
- Is 3^(2*n) = O(3^n)
  Yes, the constant in the exponent falls away and the result is O(3^n)

Big Theta with f(n) and g(n)

- f(n) = (4n^150) + (2n+1024)^400
  g(n) = 20n^400 + (n+1024)^200
  Is f(n) = theta(g(n))? or is g(n) a tight lower bound of f(n)?
  Yes. O(f(n)) = O(n^150 + n^400) vs O(n^400 + n^200)
  I would expect g(n) to be a **tight** bound if it were considered to be big Theta for f(n). g(n) appears to grow FASTER that f(n), so it would not be a tight bound.

  Another way to look at this: c1*g(n) <= f(n) <= c2*g(n)
  Which gives c1*(n^200) <= n^150 <= c2*(n^200).
  c1<1 and c2 > 1 for any N >= 1.

- f(n) = n^1.4 *4^n*
  g(n) = n^200 3.99^n
  Is f(n) = theta(g(n)), or is g(n) a tight bound of f(n)?
  O(f(n)) = 4^n. O(g(n)) = 3.99^n.
  c1*(3.99^n) <= 4^n <= c2(3.99^n)
  This is very close and I neglected the first terms is both functions, so if the n^1.4 and n^200 were included, for very large input datasets.
  c1<1
  c2>1
  n > 0.

- f(n) = 2^{log(n)}
  g(n) = n^{1024}
  Reference: 1 < logn < sqrt(n) < n < nlogn < n^2 < n^3 < ... 2^n < 3^n < n^n
  c1*(n^1024) <= 2^(log(n)) <= c2*(n^1024)
  c1 = 0 c2 > 1 n > 1

Analyze the Algorithm 1 and give a tight theta bound on the running time as a function of n. Carefully describe your justifications.

Analyze the Big O of the pseudocode - What is the tight Theta?

The code has one while loop and one For Loop with a While loop inside. While loop: N - that is, it will traverse the entire input dataset.

While Loop : N For Loop: N So this while loop inbedded with the For loop would be N2. This code would be N + N^2 or **2N^2**.
So Big O(n) would be O(n^2). Big Theta would also be Theta(n^2)

Analyze the algorithm below. What is the Big O?

The psuedo code contains:
For Loop i from 0 to n

```
    for loop from 0 to i*n<br>
```

The inner loops signifies N^2 I believe. The outer loop another N. So O(f(n)) is O^3