Name: Austin Yeh, Eric Deng

EID: ay6922, cd36549

1

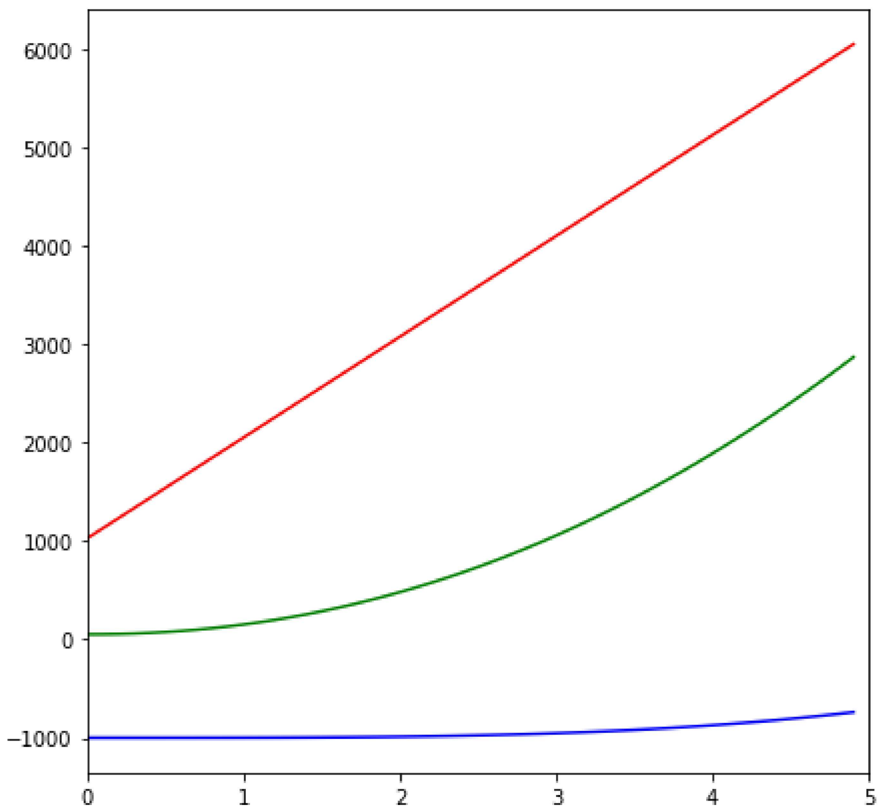
In [2]:

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

msize = 5

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,

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

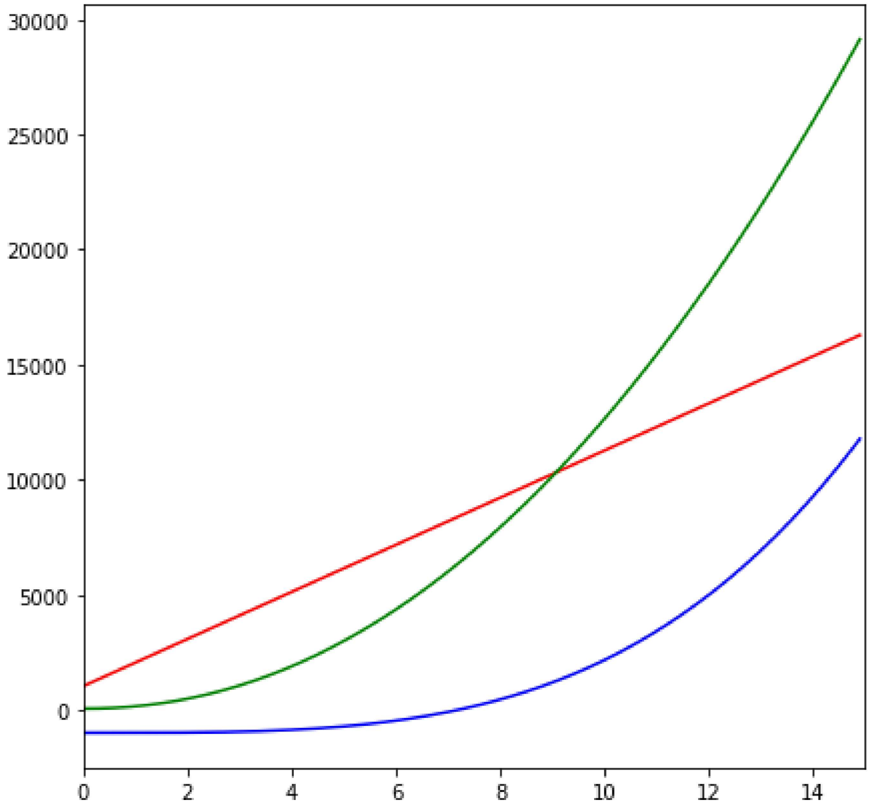


In [6]:

```
msize = 15

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,

plt.xlim(0, msize)
plt.show()
```

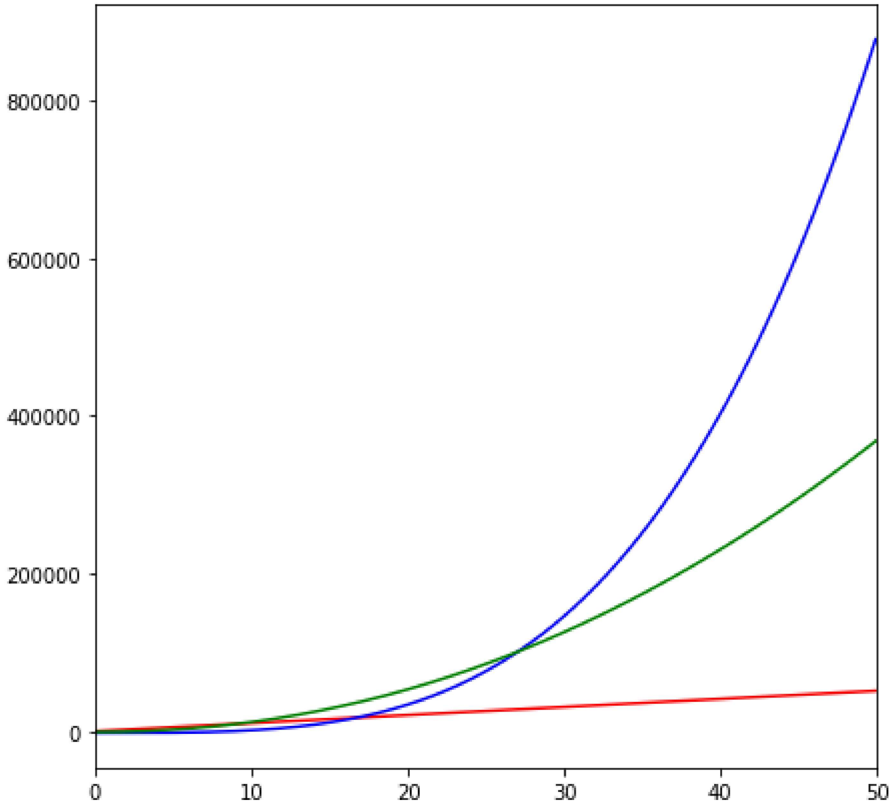


In [7]:

```
msize = 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,

plt.xlim(0, msize)
plt.show()
```



In each of the three graphs, the fastest growing functions in each interval are different. f_1 is the fastest growing function when $t < 5$; f_3 is the fastest growing function when $t < 15$; f_2 is the fastest growing function when $t < 50$.

2

$f(n) = 2^{(n+1.3)} = 2^n \times 2^{1.3}$

If we let $c = 2^{1.3} + 1$, then $0 \leq f(n) \leq c \times g(n) = c \times 2^n$

Therefore this statement is true.

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

There does not exist a constant c such that $0 \leq f(n) \leq c \times g(n) = c \times 3^n$

Therefore this statement is false.

3

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

In this euqation, $f(n) = O(g(n))$. If we let $c = 2^{400}$, then $0 \leq f(n) \leq c \times g(n)$

2. $f(n) = n^{1.4} \times 4^n$ vs. $g(n) = n^{200} \times 3.99^n$

In this euqation, $f(n) = O(g(n))$. With the base as an integer, the dominating power of $f(n)$ is 4^n and that of $g(n)$ is 3.99^n, and the with the base as n, the dominating power for both function is

$n^(1.4)$ and n^{200}. $f(n)$ has a greater growth rate on interger-base power while $g(n)$ has a great growth rate on the n-base power. It is quite hard to just use this piece of information to form a conclusion, so again we use the limit. The $lim\ f(n)/g(n)$ can be factor as the following formula: $lim\ (1.00251^n/n^{198.6})$ as n approches infinity. By using L'Hospital Rule on this limit, eventually the answer is 0, which means that $g(n)$ grows faster than $f(n)$ and proving that $f(n) = O(g(n))$.

3. $f(n) = 2^{log(n)}$ vs. $g(n) = n^{1024}$

In this euqation, $f(n) = O(g(n))$. Again, there is no similar donination of power like function 1 to let us determine directly, so we use the limit. The formula of $lim\ f(n)/g(n)$ can be factor as the following formula: $lim\ e^{ln(n)ln(2)-1024ln(n)}$ by using exponent rule. Simplify the function and we can get $lim\ e^{ln(2)-1024ln(n)}$ as n approches to infinity. Eventually with the limit we will get $infty^{-C}$, where C is a constance. This gives us the answer of 0, proving that $f(n) = O(g(n))$.

4

We know that the maximum Big O value would be from line 6 to line 12 because it is nested. The outer for loop has an iteration of n. The inner while loop, since it is iterated using i = i + j, the iteration is $O(log(n))$. Multiplying the two together, we get that the algorithm is $O(nlog(n))$

5

The inner for loop runs in $n \times (0 + 1 + 2 + ... + (n - 1))$, which is $O(n)$, and the outer for loop runs in $O(n)$, so after multiplying the two, we get that the algorithm is $O(n^3)$