```
function x = f(n)
  x = 1;
  for i = 1:n
    for j = 1:n
      x = x + 1;
```

1. Find the runtime of the algorithm mathematically (I should see summations).

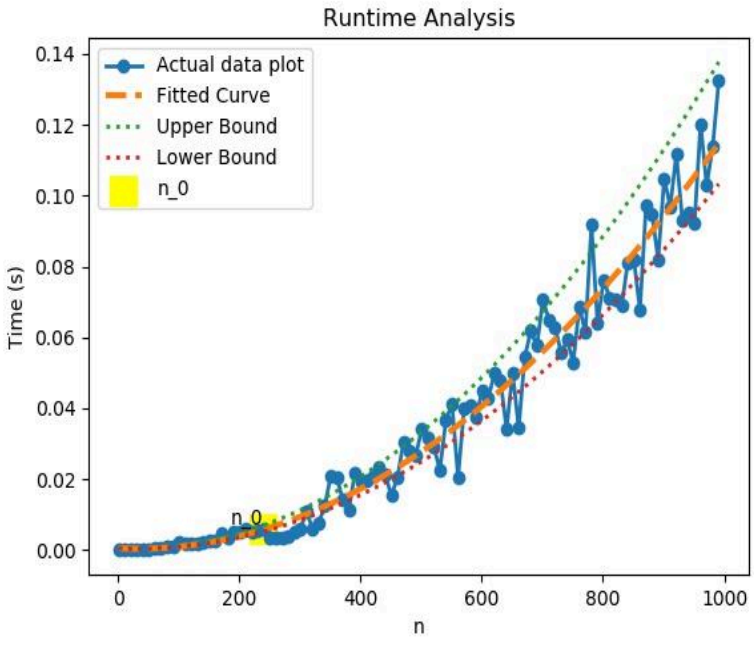
The algorithm has two for loops. The outer loop runs from 1 to n and the inner loop also runs from 1 to n. Hence the runtime of the above algorithm can be expressed as below:

$$T(n)=\sum_{i=1}^{n}\sum_{j=1}^{n} 1 \Rightarrow \sum_{i=1}^{n}(n) \Rightarrow n.\sum_{i=1}^{n} 1 = n.n = n^2$$

The runtime of the algorithm $T(n) = n^2$ and the time complexity is $O(n^2)$.

2. Time this function for various n e.g. n = 1,2,3.... You should have small values of n all the way up to large values. Plot "time" vs "n" (time on y-axis and n on x-axis). Also, fit a curve to your data, hint it's a polynomial.

Considering n from 1 to 1000 , plotting n vs time (used timeit in python) and fitting a curve for the same,



3. Find polynomials that are upper and lower bounds on your curve from #2. From this specify a big-O, a big-Omega, and what big-theta is.

Let us assume $T(n) = an^2 + bn + c$ as the is the curve fit. The upper bound and lower bound are as below:

Upper bound - $a`n^2 + b`n + c$

Lower bound - $a``n^2 + b``$

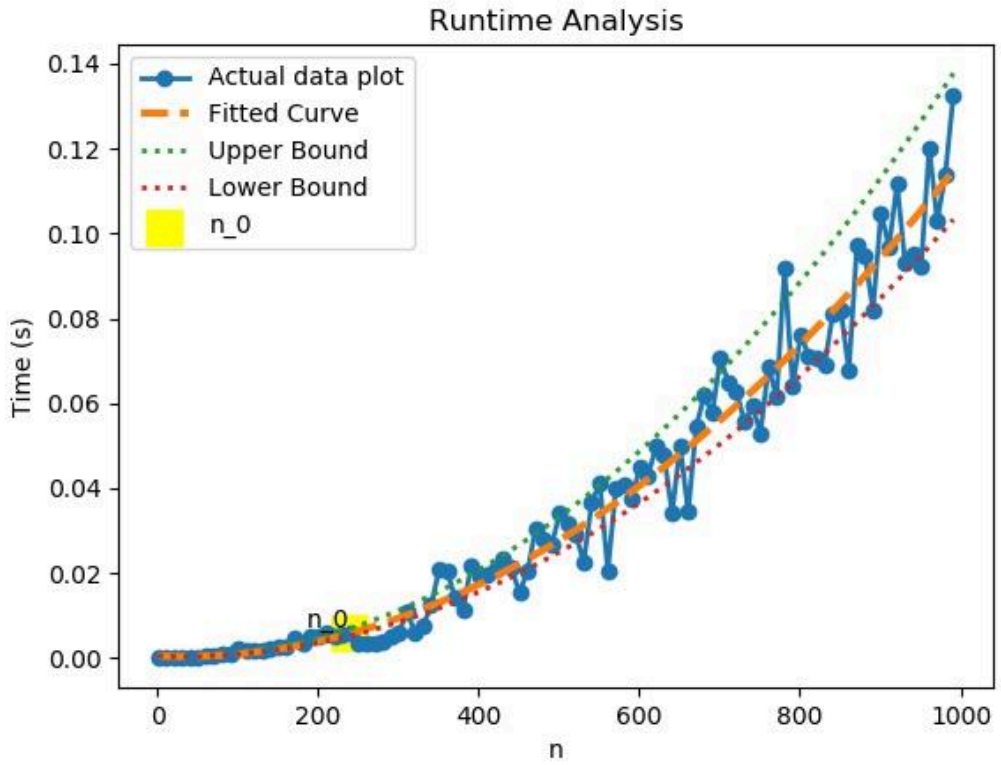
Big O Notation - $O(n^2)$

Big Ω Notation - $\Omega(n^2)$

Big Θ Notation - $\Theta(n^2)$

4. Find the approximate (eyeball it) location of "n_0" . Do this by zooming in on your plot and indicating on the plot where n_0 is and why you picked this value. Hint: I should see data that does not follow the trend of the polynomial you determined in #2.

Indicating n_0 in the plotted graph as it deviates from the trend of the polynomial.

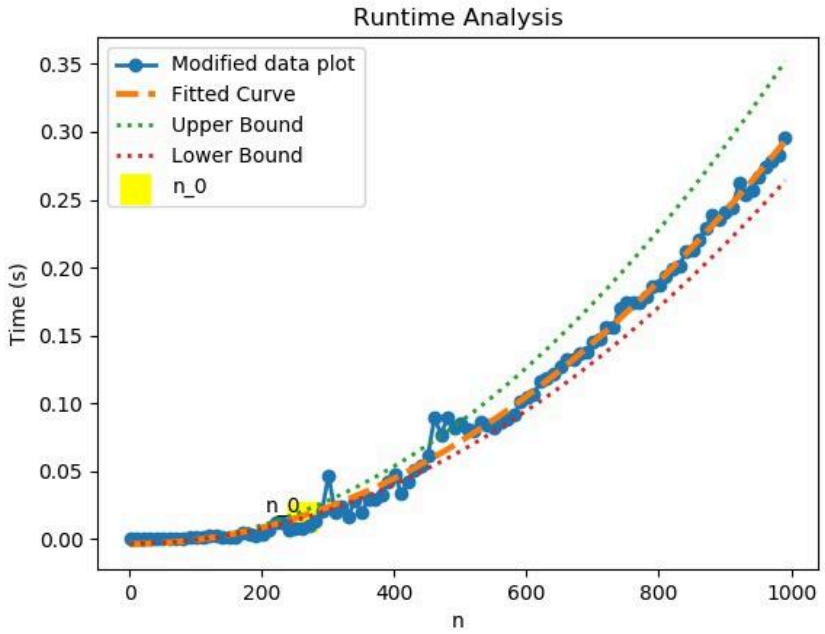


If I modified the function to be:

```
x = f(n)

  x = 1;

  y = 1;

  for i = 1:n

     for j = 1:n

        x = x + 1;

       y = i + j;
```

4. Will this increate how long it takes the algorithm to run (e.x. you are timing the function like in #2)?

Yes, the modification introduces an additional operation inside the nested loop, increasing the overall running time.



5. Will it effect your results from #1?

The modification changes the number of operations in each iteration but doesn't change the overall time complexity, which remains $O(n^2)$.