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

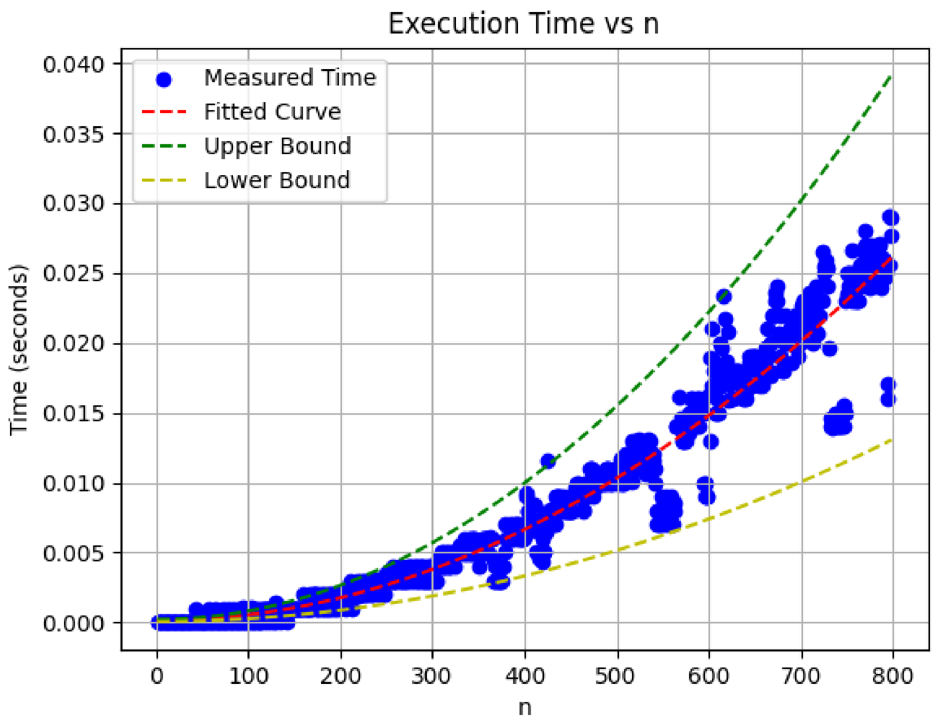
Initially, there are two nested for loops which starts from i = 1 to n and j = 1 to n.
The critical operation inside the loop is x = x +1, which executes each time for the combination of i and j. So, the total execution of x = x +1 will be

$$\sum_{i=1}^{n} \sum_{j=1}^{n} 1 = \sum_{i=1}^{n} n = n * n = n^2$$

Solving, the nested loops give n^2.

Therefore, the runtime of the function becomes $O(n^2)$.

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.



The function is plotted against time vs the number of executions.
Upper bound and lower bound curve are based on the assumption of 50% away from the fitted curve.

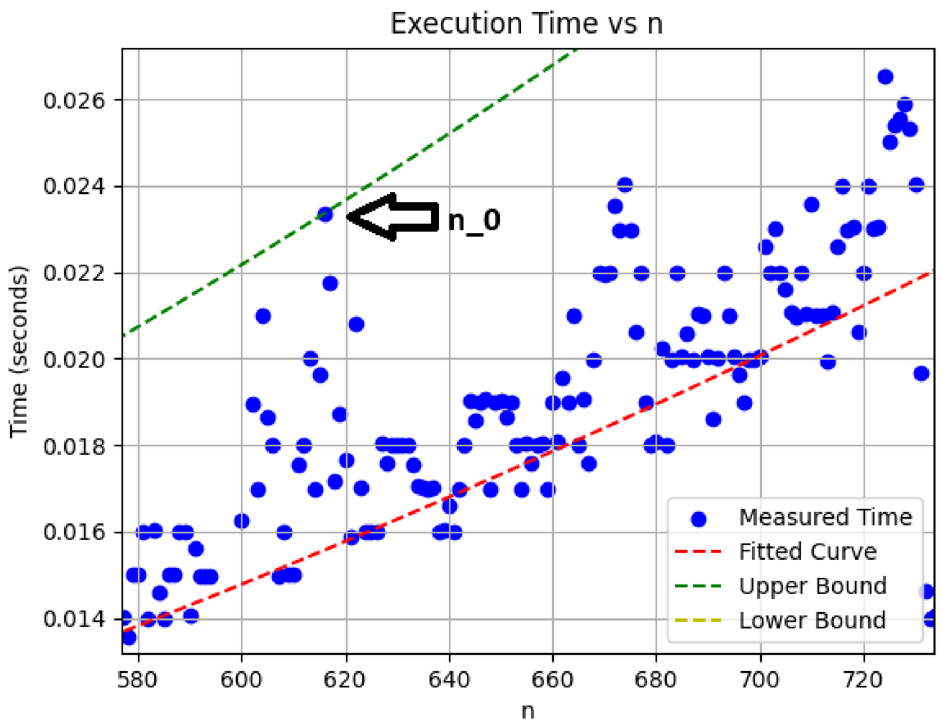
3. Find polynomials that are upper and lower bounds on your curve from #2. From this specify a big-O, a big-Omega, and what big-theta is.

Based on the curve we see, the curve resemblance in the quadratic form.
Therefore, we can assure that the runtime of the function be O(n^2).
Big-O upper bound=> based on the runtime of the function, it does not take more than n^2 operation time giving it as O(n^2).

Big-Omega=> based on the runtime of the function, it will at least take n^2 operation time giving it as O(n^2)

Big-Theta=> Since the function takes no more or at least n^2 of operation time giving it as O(n^2)

4. Find the approximate (eye ball 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.



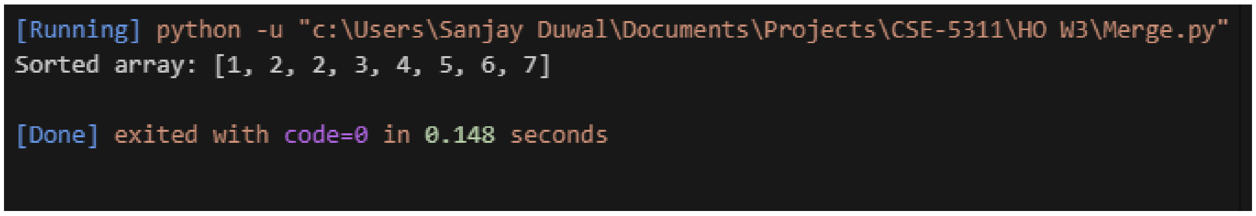
After zooming and observing we can approximate that the n_0 somewhere around 618, which is far from the fitted curve and follows less degree of the polynomial.

4. Will this increase how long it takes the algorithm to run (e.x. you are timing the function like in #2)?
Yes, it will increase the time by some amount but not significantly. It is just adding an additional operation and the operation takes O(1) of time complexity.
No, as the time complexity remains O(n^2).

5. Will it effect your results from #1?

No, it will not effect the result from #1 because the time complexity remains same which is O(n^2). The additional constant time O(1) does not make difference in it.

6. Implement merge sort, upload your code to github and show/test it on the array [5,2,4,7,1,3,2,6].



```
[Running] python -u "c:\Users\Sanjay Duwal\Documents\Projects\CSE-5311\HO W3\Merge.py"
Sorted array: [1, 2, 2, 3, 4, 5, 6, 7]

[Done] exited with code=0 in 0.148 seconds
```