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

To analyze the runtime of the given algorithm, we can count the number of iterations it performs. The inner loop runs n times for each iteration of the outer loop, and there are a total of n iterations of the outer loop. Therefore, the total number of iterations is given by the sum:

A white board with purple writing on it

Description automatically generated

1. 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.
2. 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.
3. 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.

* The answer is in the HandsOn.ipynb code file

A graph of data on a white grid

Description automatically generated

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)?

5. Will it effect your results from #1?

🡪 The answer is in the HandsOn.ipynb code file

A graph of data on a white grid

Description automatically generated