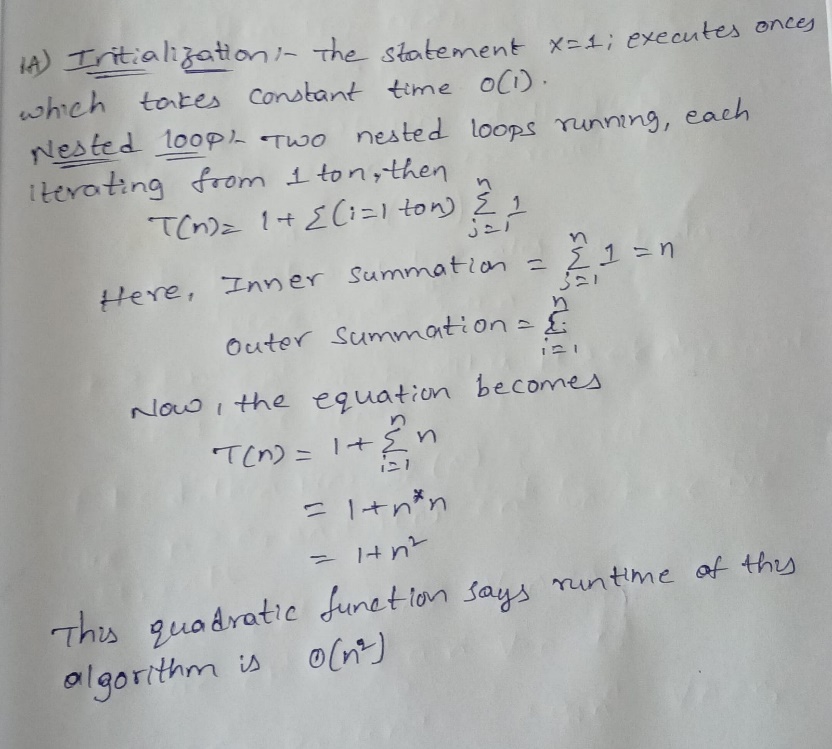
**Hands-on 3**

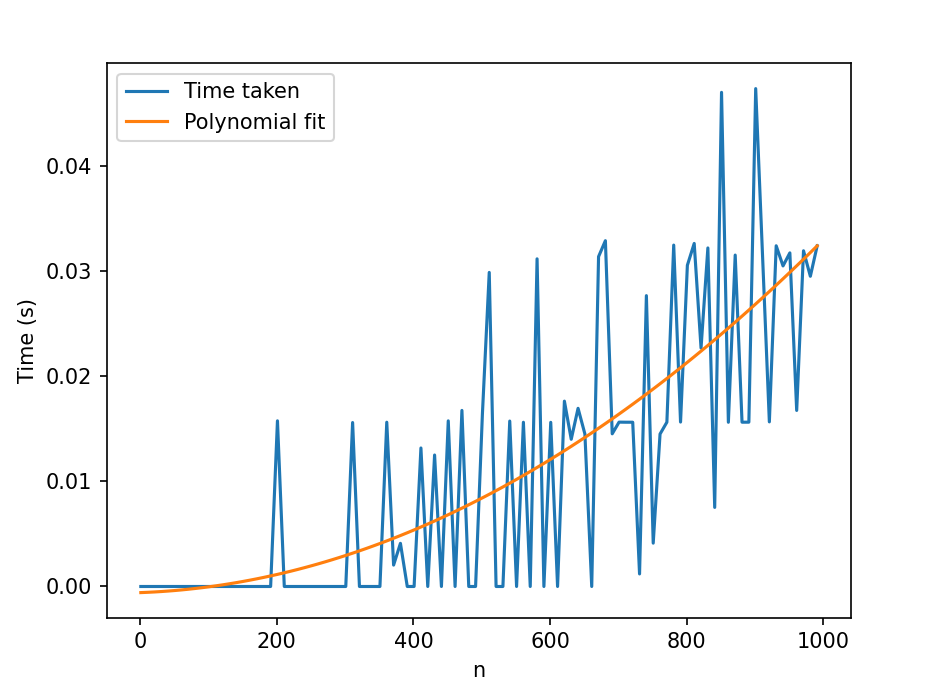
1. **Find the runtime of the algorithm mathematically.**

****

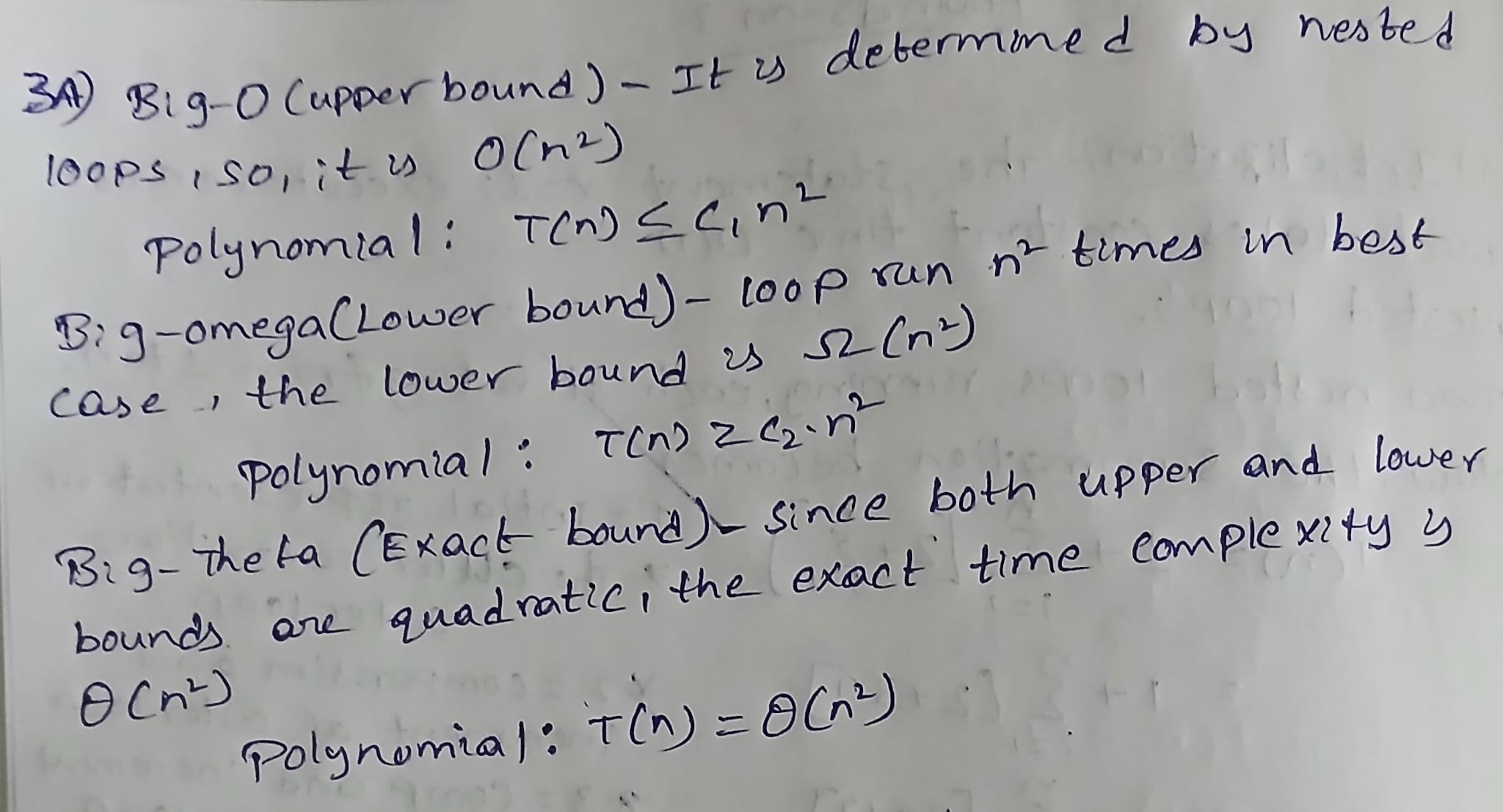
1. **Time this function for various n e.g. n = 1,2,3.... You should have small values of n all the way up to large values. Plot "time" vs "n" (time on y-axis and n on x-axis).**

Refer second code to find the python code in the repository

Output:

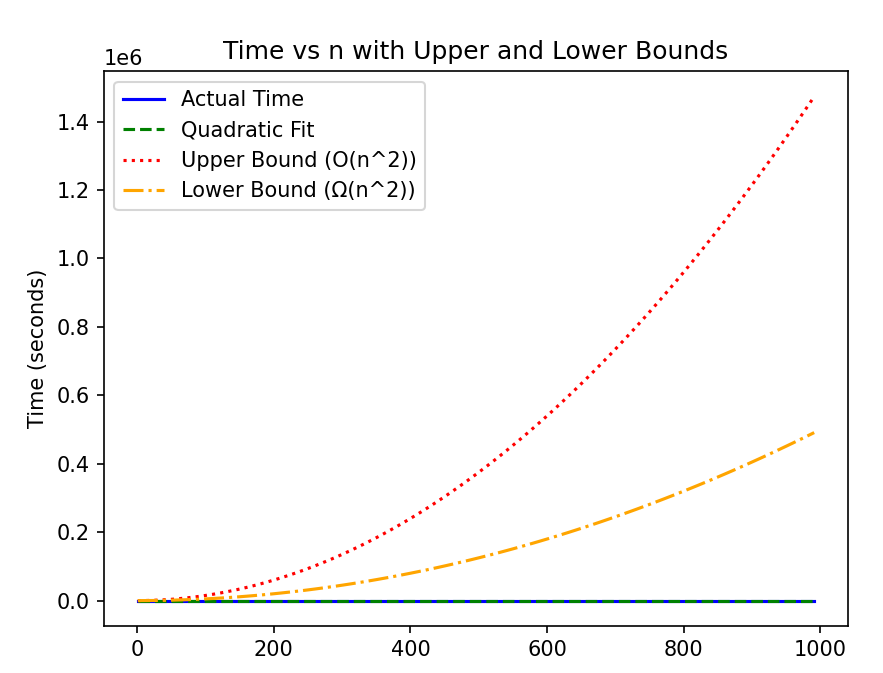


1. **Find polynomials that are upper and lower bounds on your curve from #2. From this specify a big-O, a big-Omega, and what big-theta is.**



Refer third code to find the python code which plots upper, lower and actual fit.

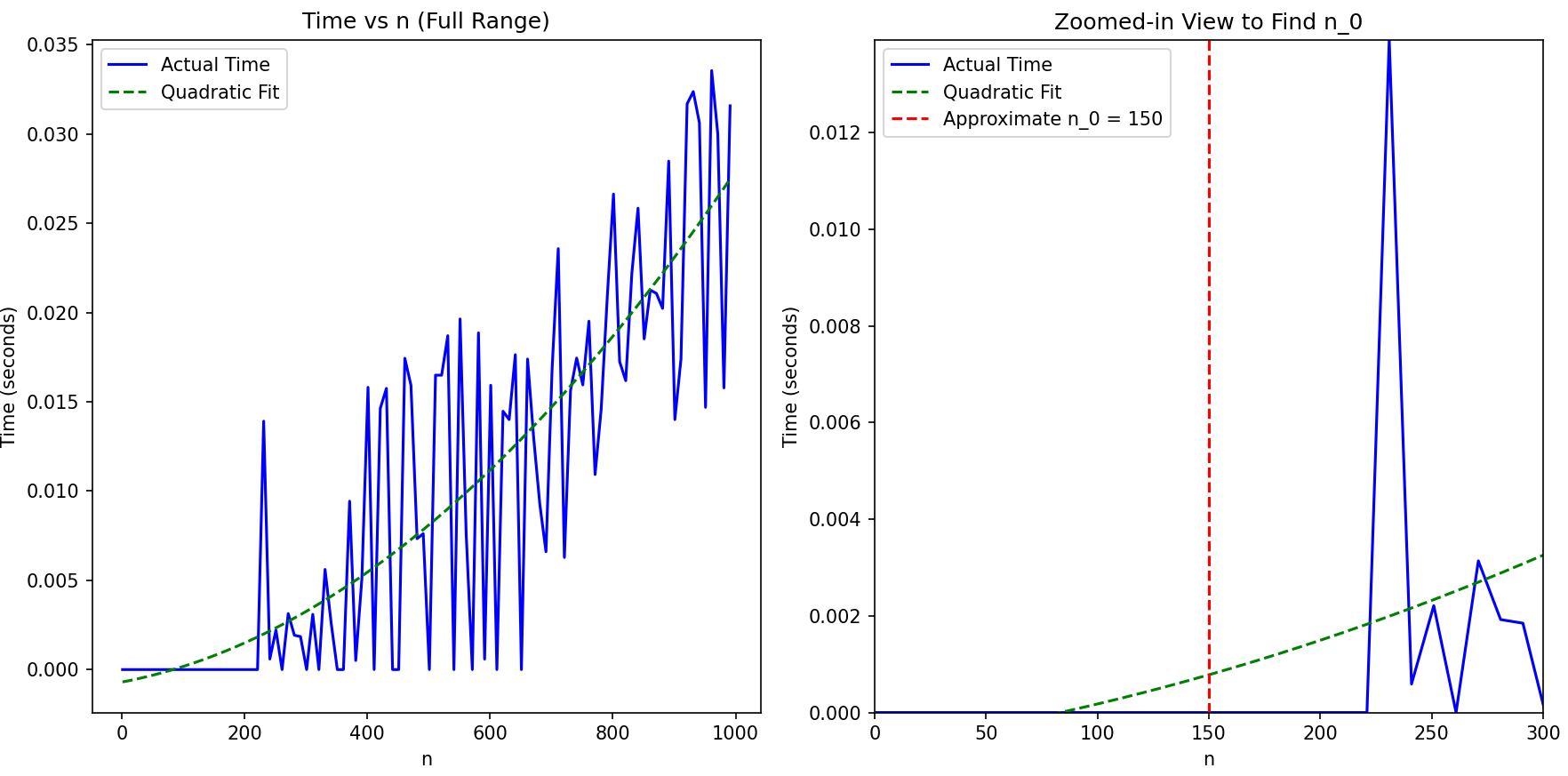
**Output**:

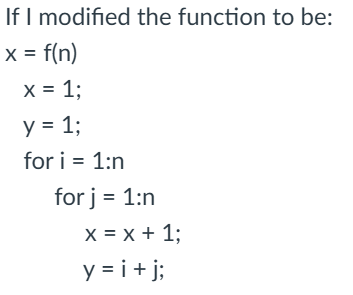


1. **Find the approximate (eye ball it) location of "n\_0" . Do this by zooming in on your plot and indicating on the plot where n\_0 is and why you picked this value. Hint: I should see data that does not follow the trend of the polynomial you determined in #2.**

Refer fourth code to find the python code.

**Output**:





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

Yes, but only slightly. The added operation y = i + j is an O(1) operation (constant time), so it will slightly increase the total time for each iteration. However, the overall time complexity will still remain O(n power 2), as the added operation doesn’t affect the order of growth.

1. **Will it effect your results from #1?**

No, Overall time complexity will remain same O(n2), the number of steps in the function increased, as observed only change is in the constant value and not the structure of T(n).

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

Refer to the seventh code for the python code.

**Output**:

