Nama : Nadya Ayu Widya

NIM : L200180099

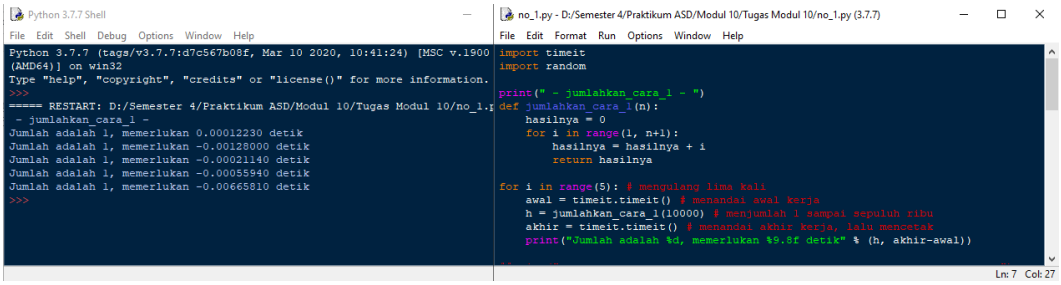
Kelas : D

TUGAS PRAKTIKUM ASD

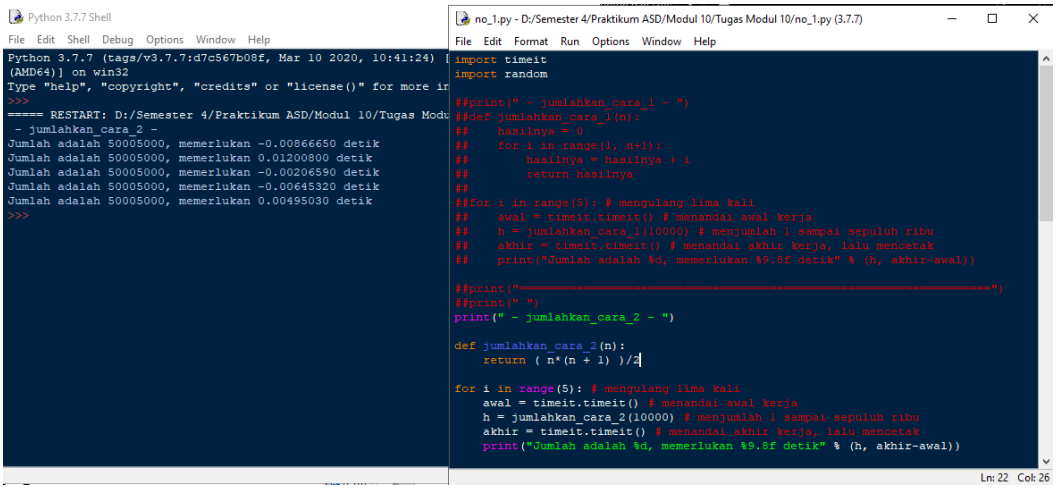
MODUL 10

Analisis Algoritma

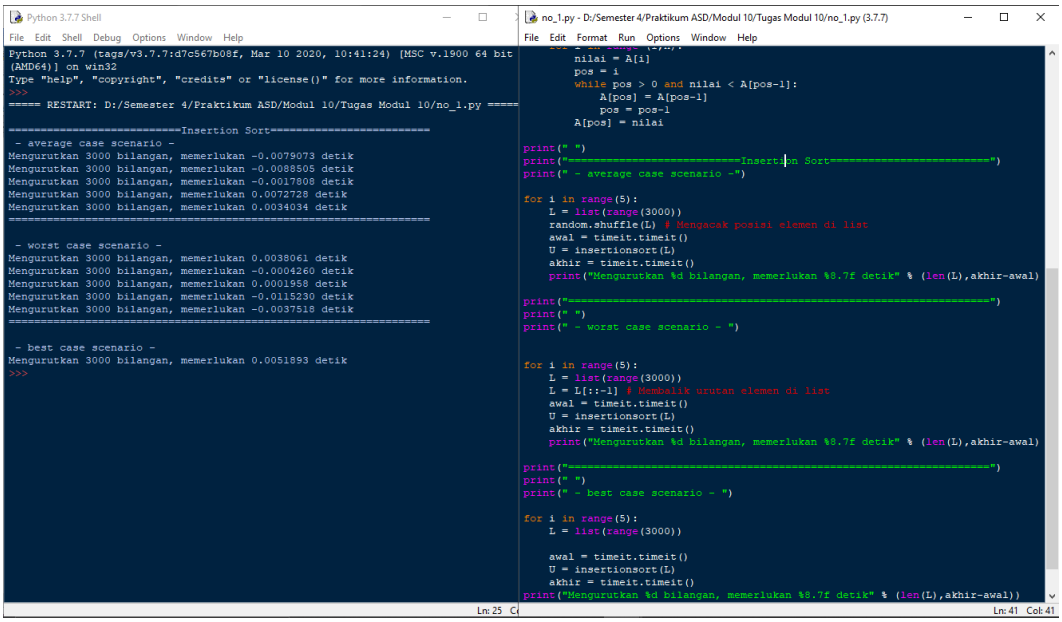
1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul timeit, yakni
   a Jumlahkan_cara_1



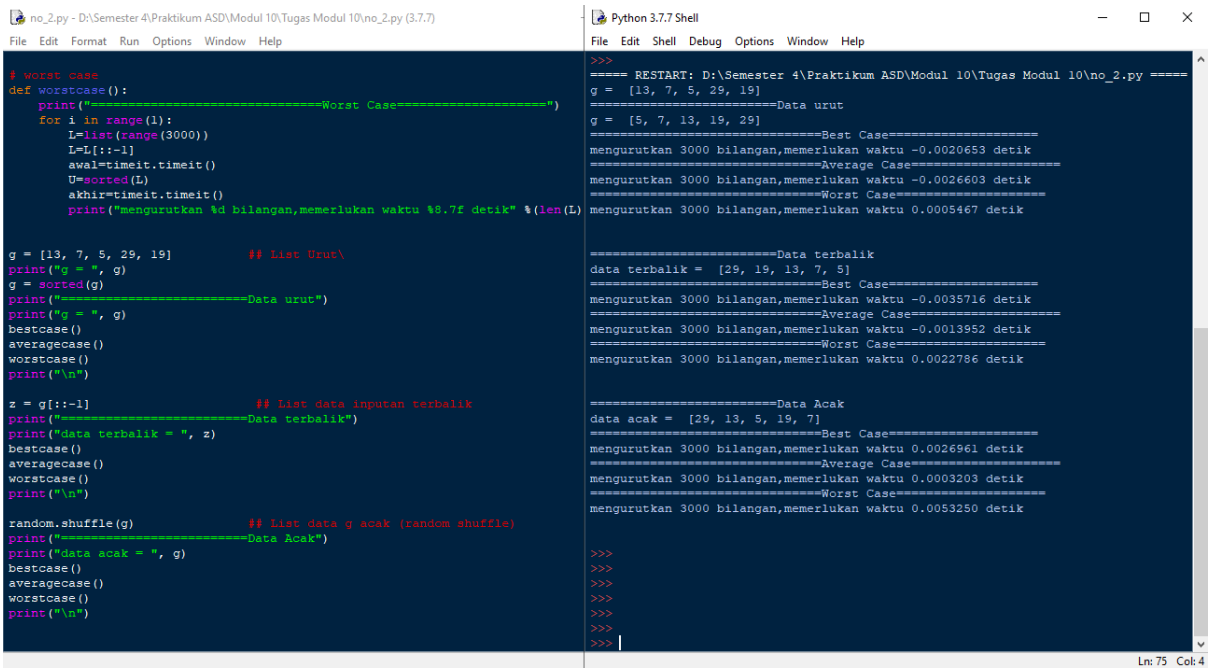
   b Jumlahkan_cara_2



   c insertionSort

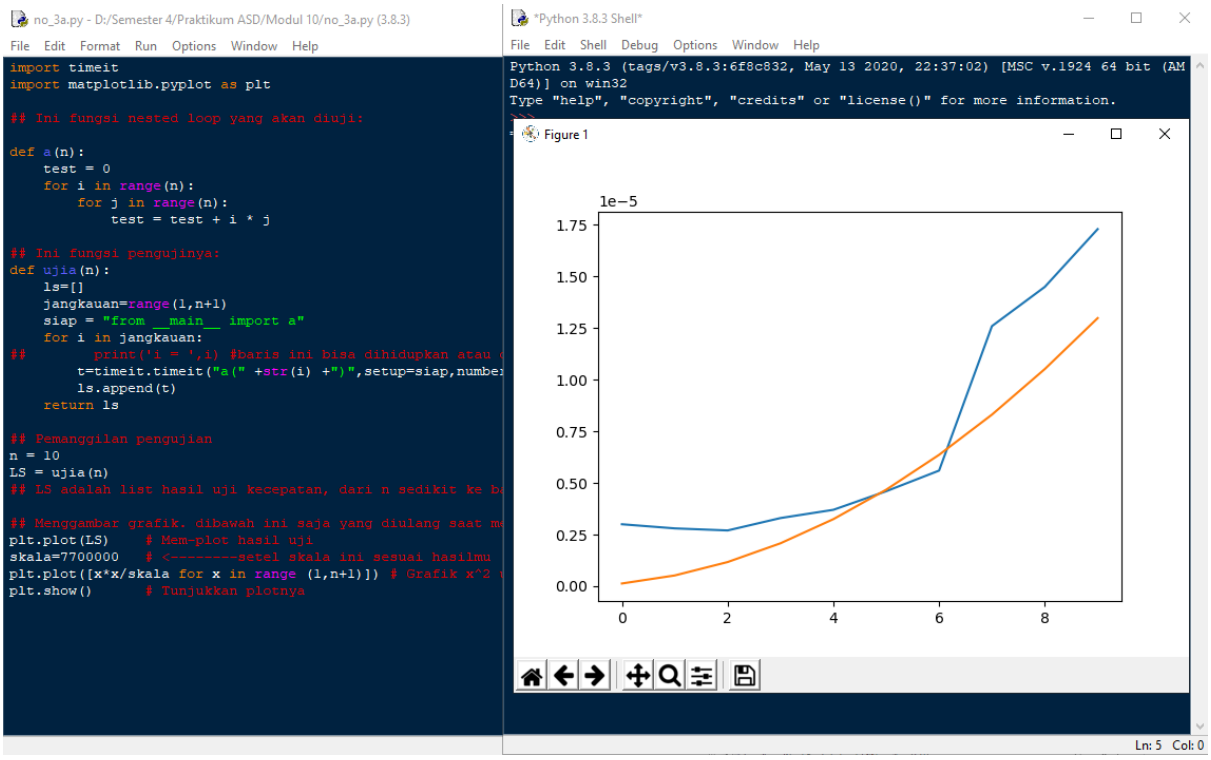


2. Python mempunyai perintah untuk mengurutkan suatu list yang memanfaatkan algoritma Timsort. Jika g adalah suatu list berisi bilangan, maka g.sort() kan mengurutkannya. Perintah yang lain sorted() mengurutkan list dan mengembalikan sebuah list baru yang sudah urut. Selidikilah fungsi sorted() ini menggunakan timeit:
    - Apakah yang merupakan best case dan average case bagi sorted( ) ?
    - confirm bahwa data input urutan terbalik bukan kasus terburuk bagi sorted( ). Bahkan dia lebih cepat dalam mengurutkannya daripada data input random



    - Dapat dibuktikan bahwa data dengan inputan terbalik bukan kasus buruk bagi sorted( ). Bahkan dia lebih cepat dalam mengurutkannya daripada data random

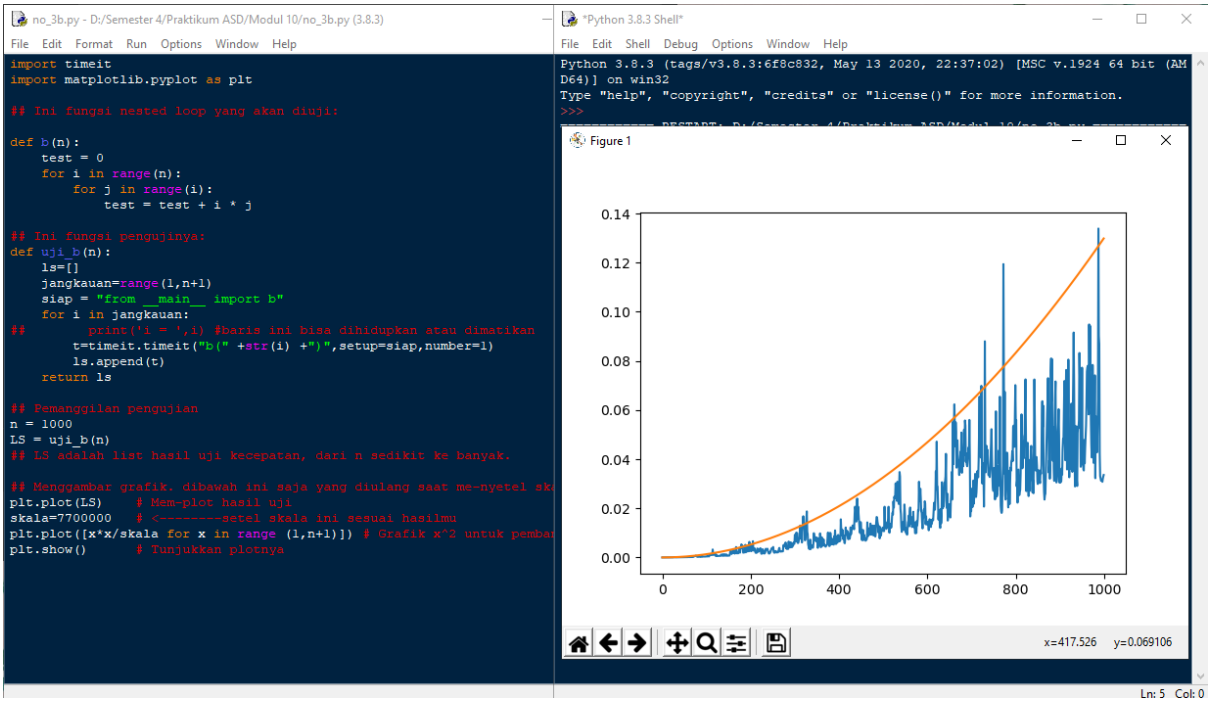
3. Untuk tiap kode berikut, tentukan running time-nya O(1), O(log n), O(n), O(n log n), $O(n^2)$ atau $O(n^3)$ atau yang lain. Untuk memulai analisis, ambil suatu nilai n tertentu lalu ikuti apa yang terjadi di kode itu
    a Loop di dalam loop keduanya sebanyak n:



T(n) = c1+n*(n)

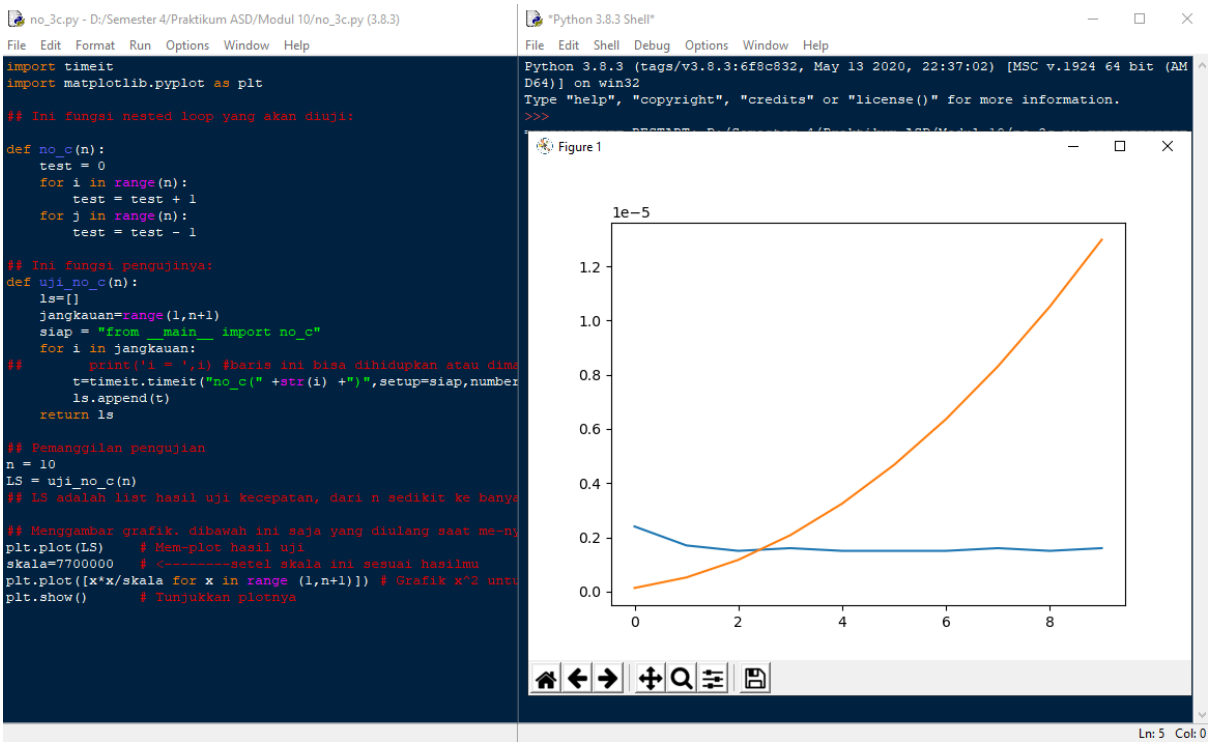
T(n) = c1+n^2
O(n) ≈ n^2
O(n^2)

b Loop di dalam loop yang dalam bergantung nilai i loop luar:



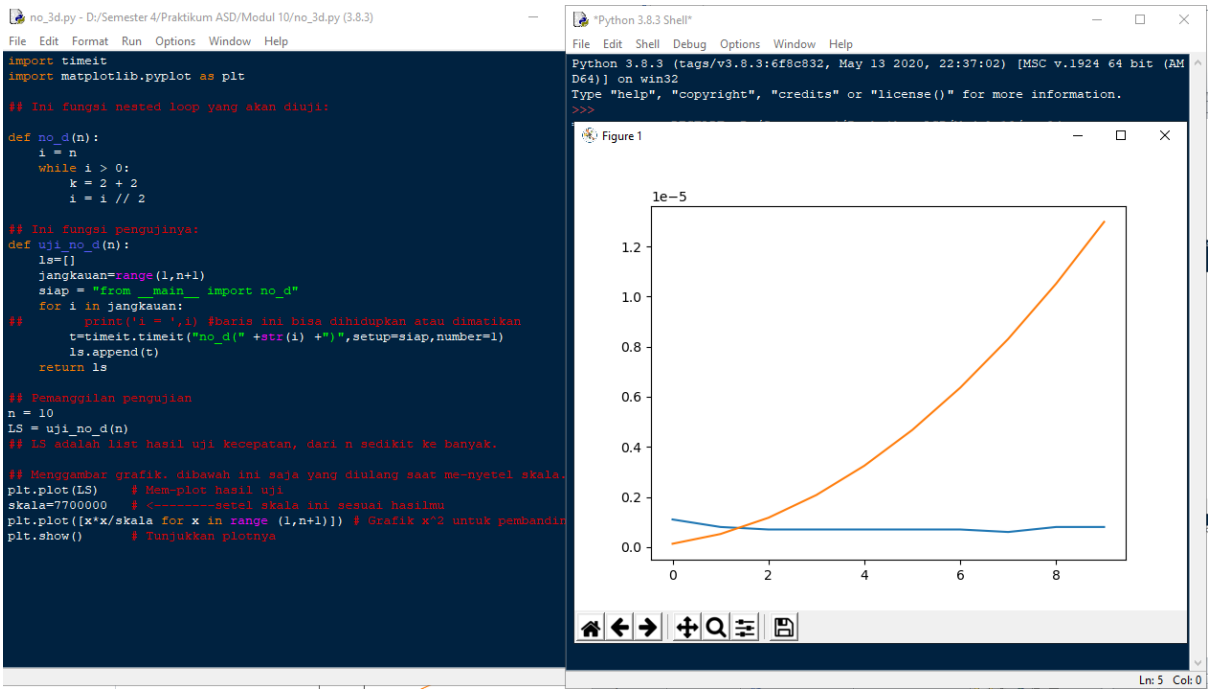
T(n) = c1+log n
T(n) = c1+log n
O(n) ≈ log n
O(log n)

c Dua loop terpisah



T(n) = c1(n) + c2(n)
T(n) = n + n
O(n) ≈ n
O(n)

d While loop yang dipangkas sepatuh tiap putaran

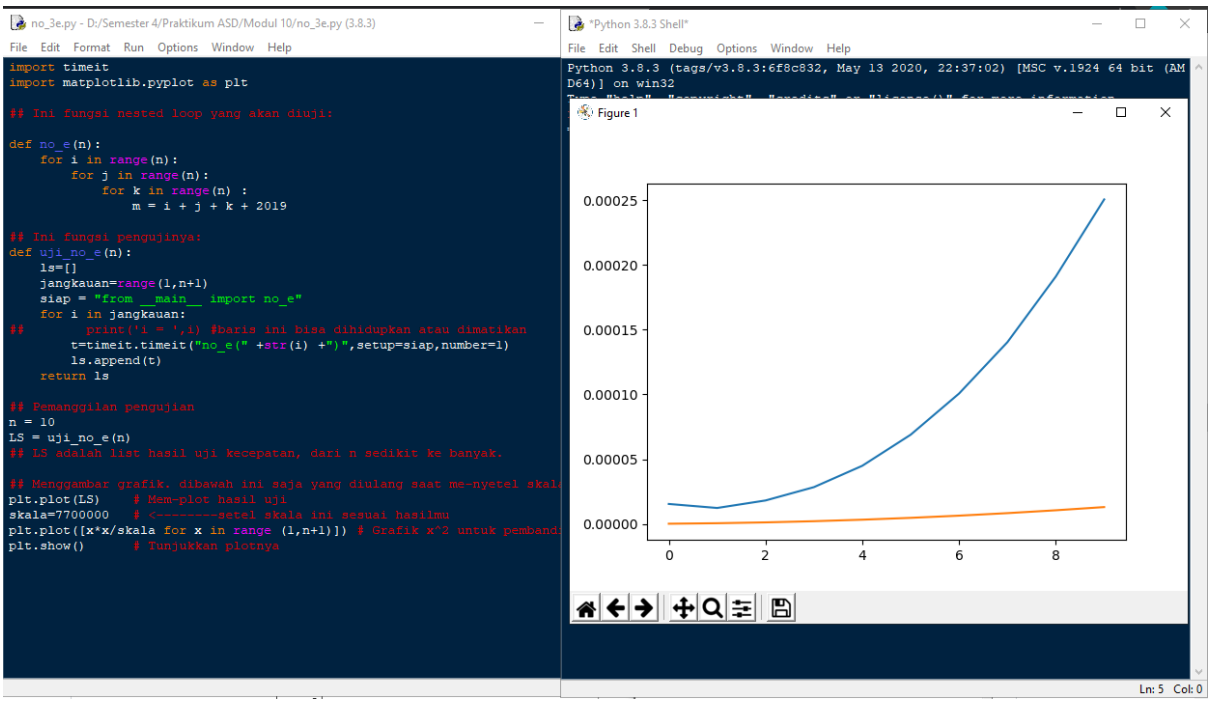


T(n) = c1(1) + c2(1)

O(n) ≈ 1

O(1)

e Loop in a loop in a loop, ketigaya sebanyak n



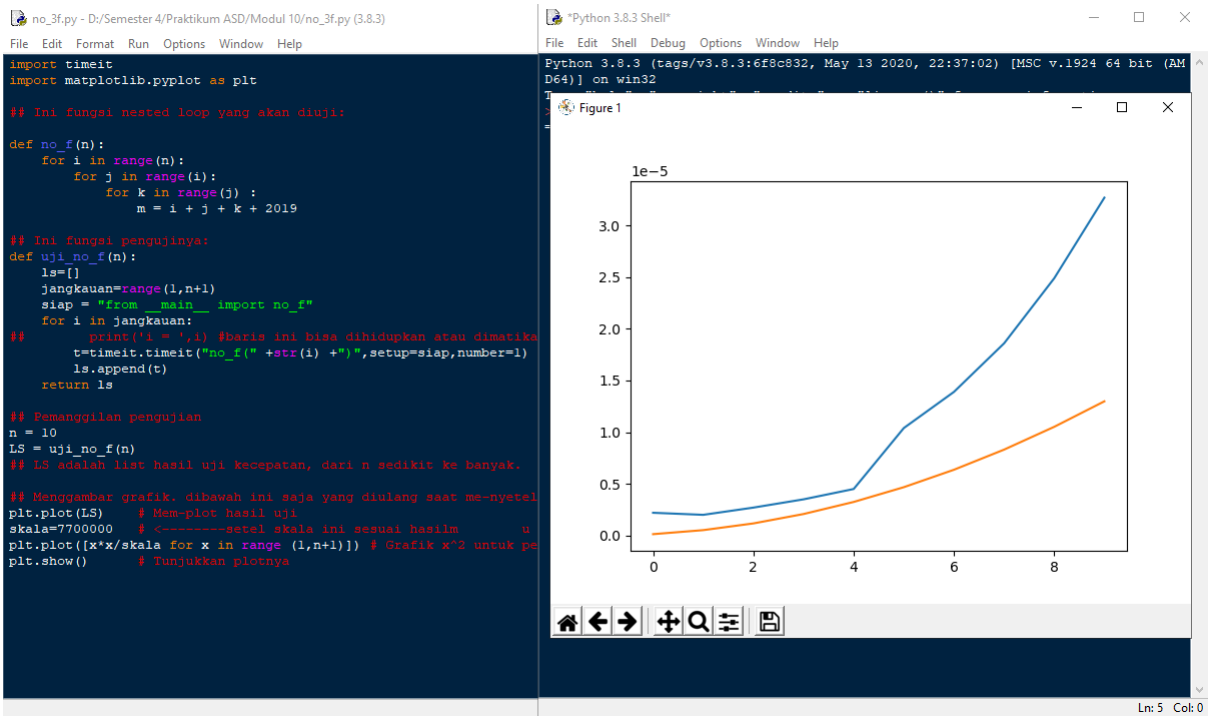
T(n) = n*(n*(n))

T(n) = n^3

O(n) ≈ n^3

O(n^3)

f Loop in a loop, dengan loop dalam sebanyak nilai loop luar terdekat



T(n) = c1(n) + c2(n) + c3(n)

T(n) = n + n + n

O(n) ≈ n

O(n)

g Fungsi ini

O(n log n)

4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat

$\log_4 n < 10\log_2 n < n \log_2 n < 2^{\log 2n} < 5n^2 < n^3 < 12n^6 < 4^n$

5. Tentukan O(.) dari fungi-fungsi berikut yang mewakili banyaknya langkah yang diperlakukan untuk beberapa algoritma

a T(n) = n^2 + 32n + 8 = O(n^2)

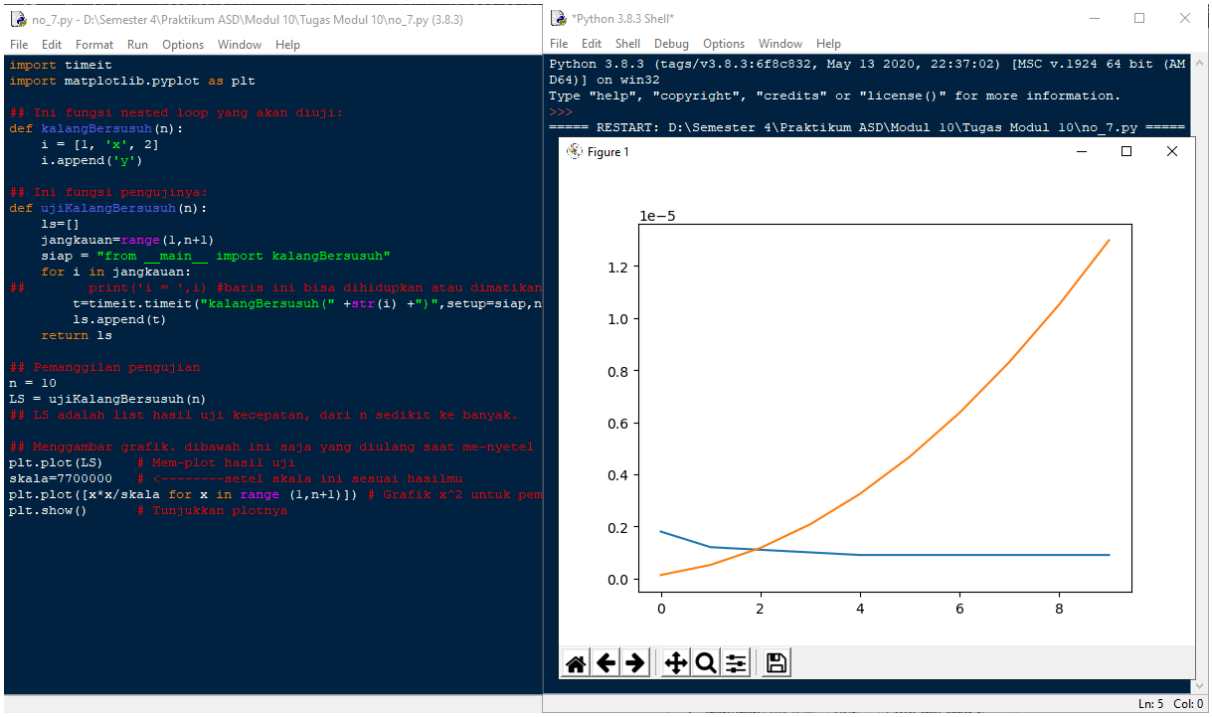
b T(n) = 87n + 8n = O(n)

c	T(n)	= 4n + 5n log n + 102	= O(n log n)
d	T(n)	= log n + 3n^2 + 88	= O(n^2)
e	T(n)	= 3(2^n) + n^2 + 647	= O(2^n)
f	T(n, k)	= kn + log k	= O(kn)
g	T(n, k)	= 8n + k log n + 800	= O(n)
h	T(n, k)	= 100kn + n	= O(kn)

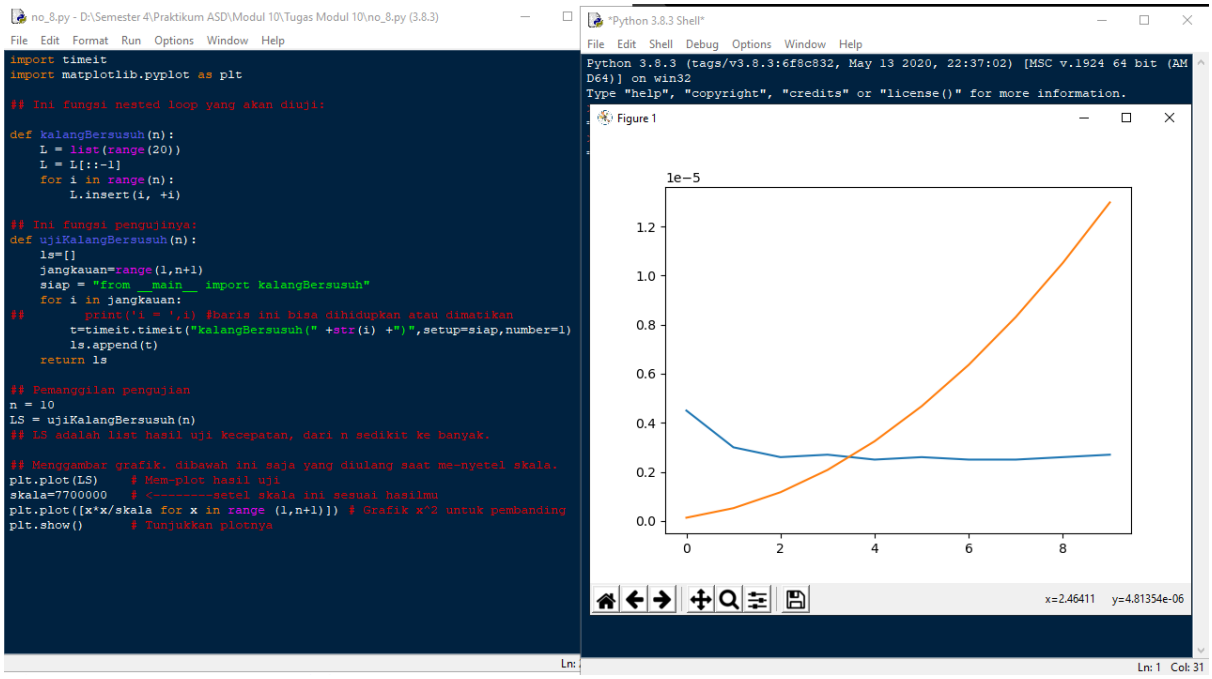
6. Carilah di internet, kompelsitas metode pada object list di python.
   - Google python list method complexity. Lihat juga bagian “Images” –nya
   - Kunjungi https://wiki.python.org/moin/TimeComplexity

Operation	Average Case	Amortized Worst Case
Copy	O(n)	O(n)
Append[1]	O(1)	O(1)
Pop last	O(1)	O(1)
Pop intermediate	O(k)	O(k)
Insert	O(n)	O(n)
Get Item	O(1)	O(1)
Set Item	O(1)	O(1)
Delete Item	O(n)	O(n)
Iteration	O(n)	O(n)
Get Slice	O(k)	O(k)
Del Slice	O(n)	O(n)
Set Slice	O(k+n)	O(k+n)
Extend[1]	O(k)	O(k)
Sort	O(n log n)	O(n log n)
Multiply	O(nk)	O(nk)
x in s	O(n)	
min(s), max(s)	O(n)	
Get Length	O(1)	O(1)

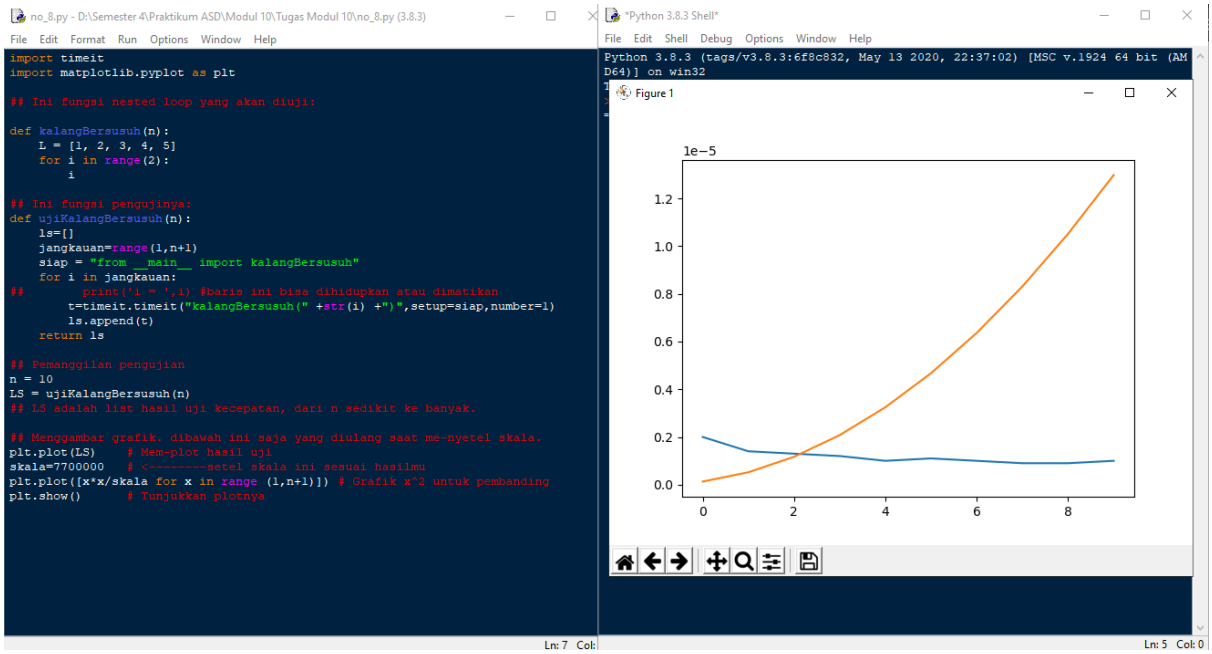
7. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode append() adalah O(1). Gunakan timeit dan matplotib seperti sebelumnya.



8. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode insert() adalah O(n). Gunakan timeit dan matplotib seperti sebelumnya.



9. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah suatu nilai berada di suatu list mempunya kompleksitas O(n). Gunakan timeit dan matplotib seperti sebelumnya.



10. Carilah di internet, kompelsitas metode pada object dict di python.

Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[2]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete Item	O(1)	O(n)
Iteration[2]	O(n)	O(n)

11. Selain notasi big-O O(.) ada pula notasi big-Theta Θ(.) dan notasi big-Omega Ω (.)
Apakah beda diantara ketiganya?
    - Big O dilambangkan dengan notasi **O(...)** merupakan keadaan terburuk (worst case). Kinerja seubuah algoritma biasanya diukur menggunakan patokan keadaan Big-O ini. Merupakan notasi asymptotic untuk batas fungsi dari atas dan bawah dengan Berperilaku mirip dengan ≤ operator untuk tingkat pertumbuhan.
    - Big Theta dilmbangkan dengan notasi **Θ(...)** merupakan notasi asymptotic untuk batas atas dan bawah dengan keadaan terbaik (best case). Menyatakan persamaan pada pertumbuhan f (n) hingga faktor konstan (lebih lanjut tentang ini nanti). Berperilaku mirip dengan = operator untuk tingkat pertumbuhan
    - Big Omega dilambangkan dengan notasi **Ω(...)** merupakan notasi asymptotic untuk batas bawah dengan keadaan rata-rata(average case) yang berperilaku mirip dengan ≥operator untuk tingkat pertumbuhan.

12. Apa yang dimaksud dengan amortized analysis dalam analisis algoritma?
Jawab: Amortized analysis adalah metode untuk menganalisis kompleksitas algoritma yang diberikan, atau berapa banyak resource nya terutama waktu atau memori yang diperlukan untuk mengeksekusi. Dapat ditunjukkan dengan waktu rata-rata yang diperlukan unyuk melakukan satu urutan operasi pada struktur data terhadap keseluruhan operasi yang dilakukan.