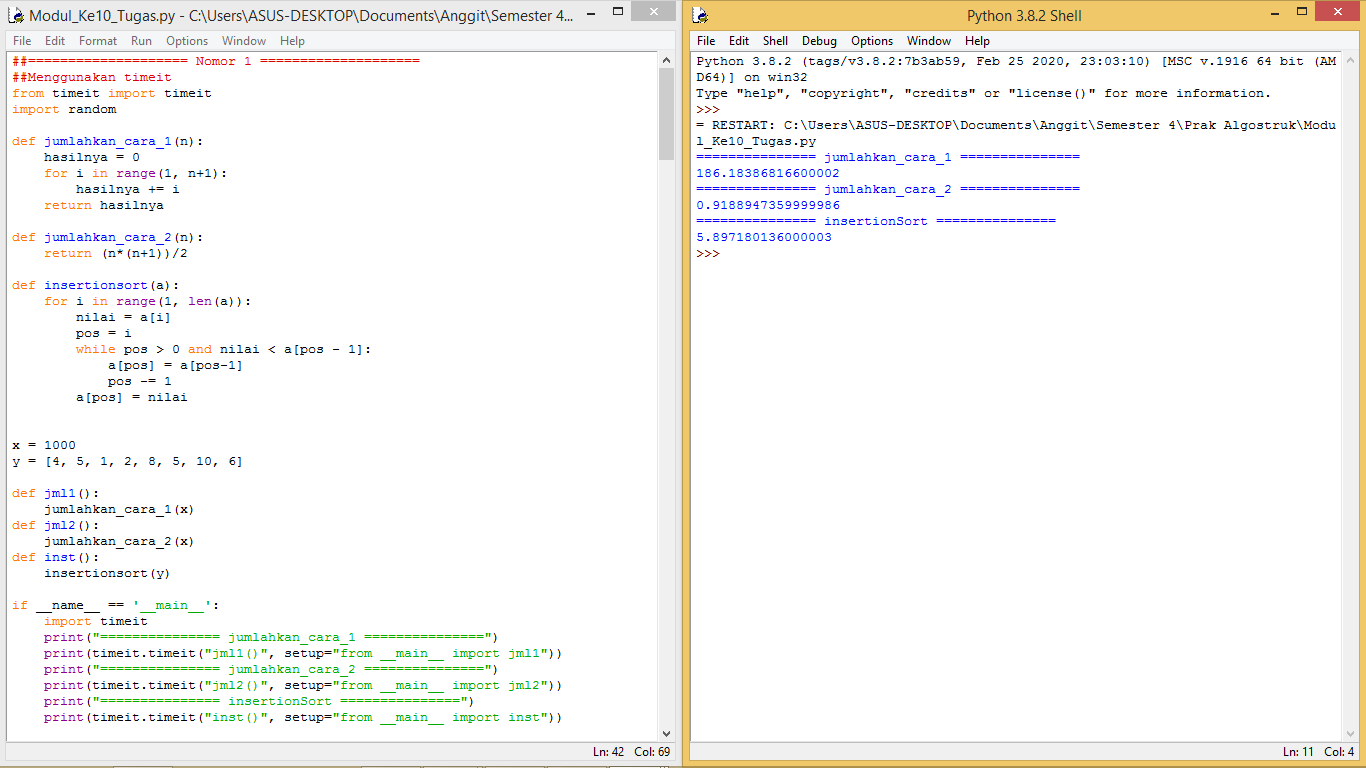
Nama : Anggit Astriani

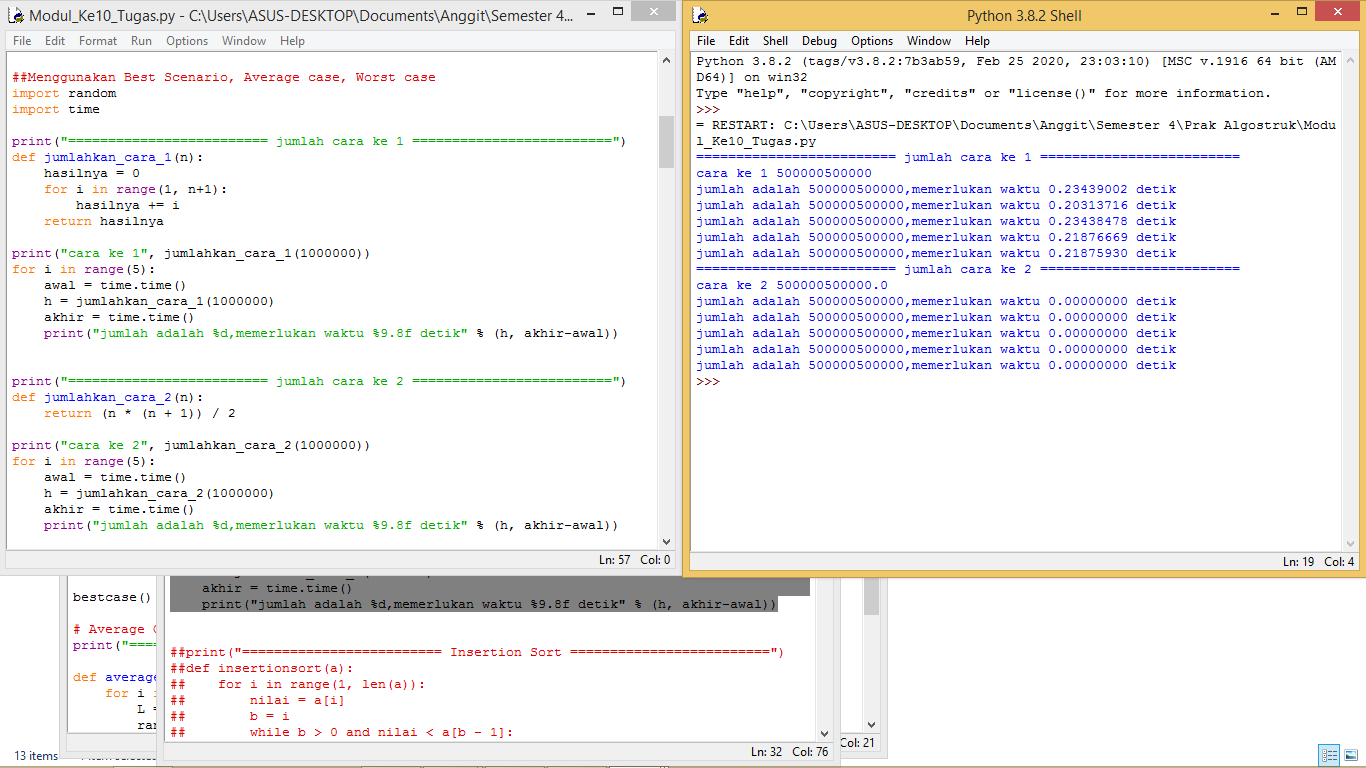
NIM : L200180111

Kelas : C

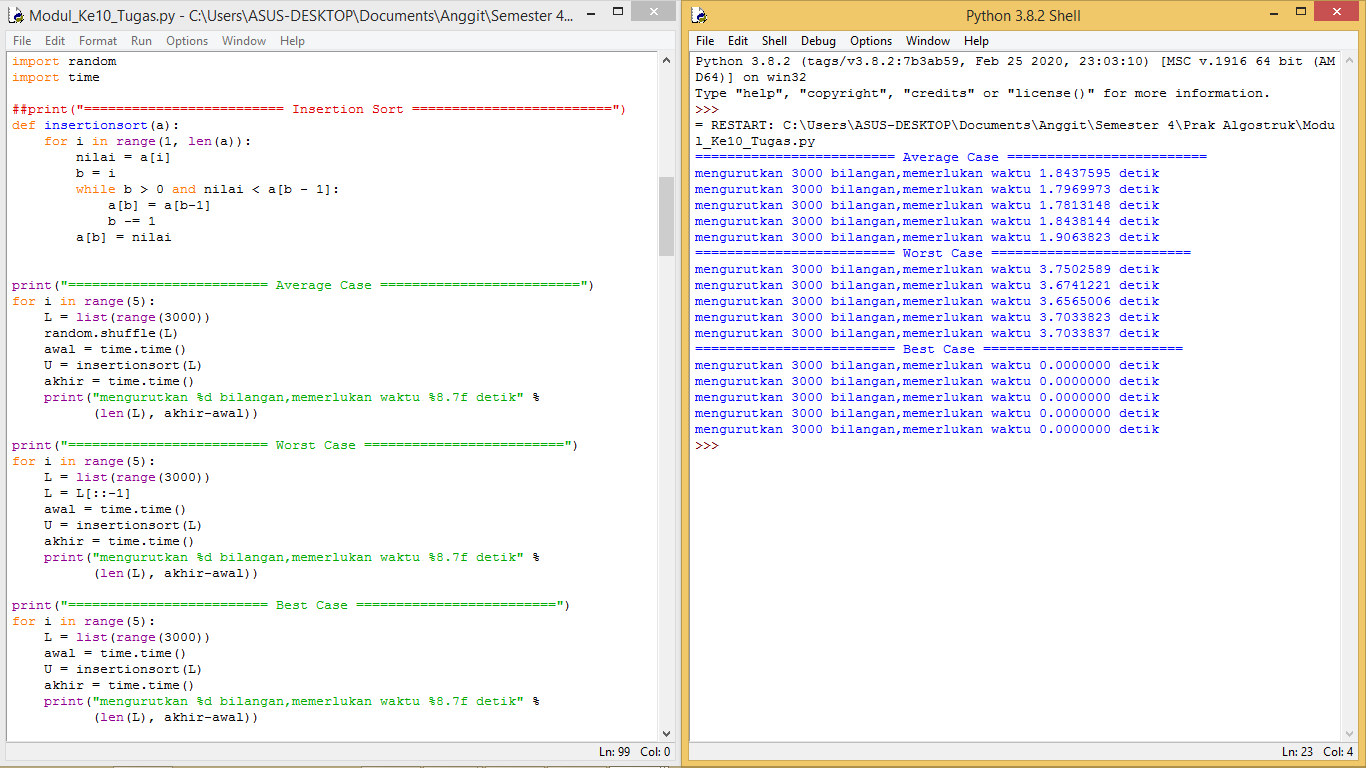
**MODUL 10**

1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul **timeit**, yakni :
2. jumlahkan\_cara\_1
3. jumlahkan\_cara\_2
4. insertionSort





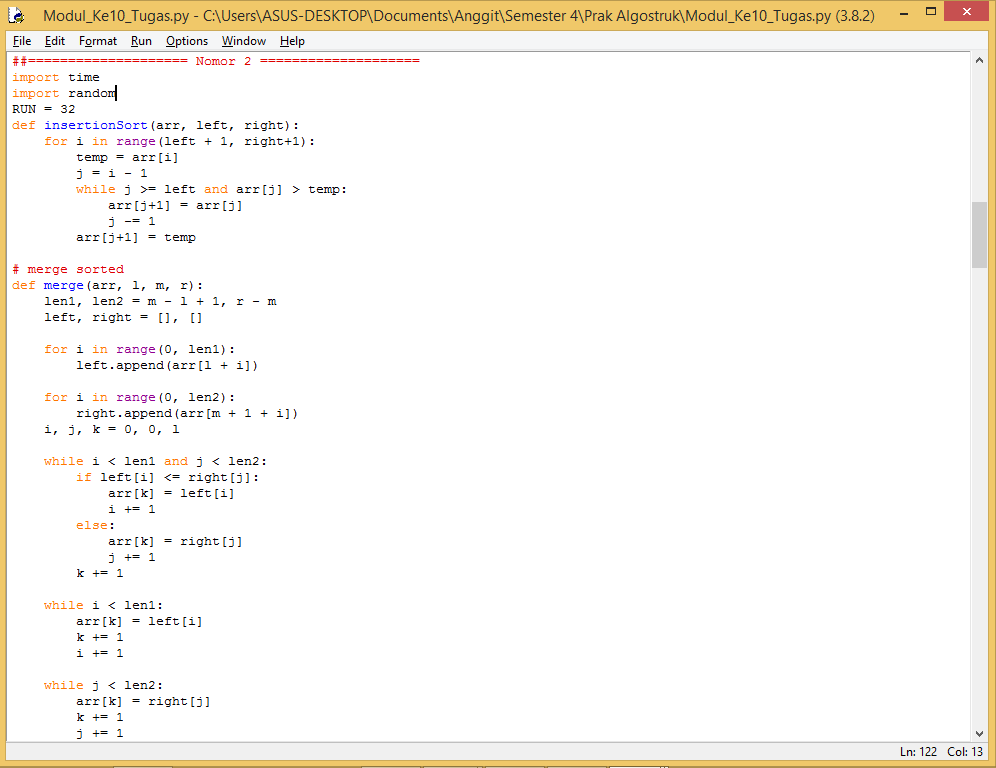
untuk insertionSort, kerjakan ketiga kasusnya

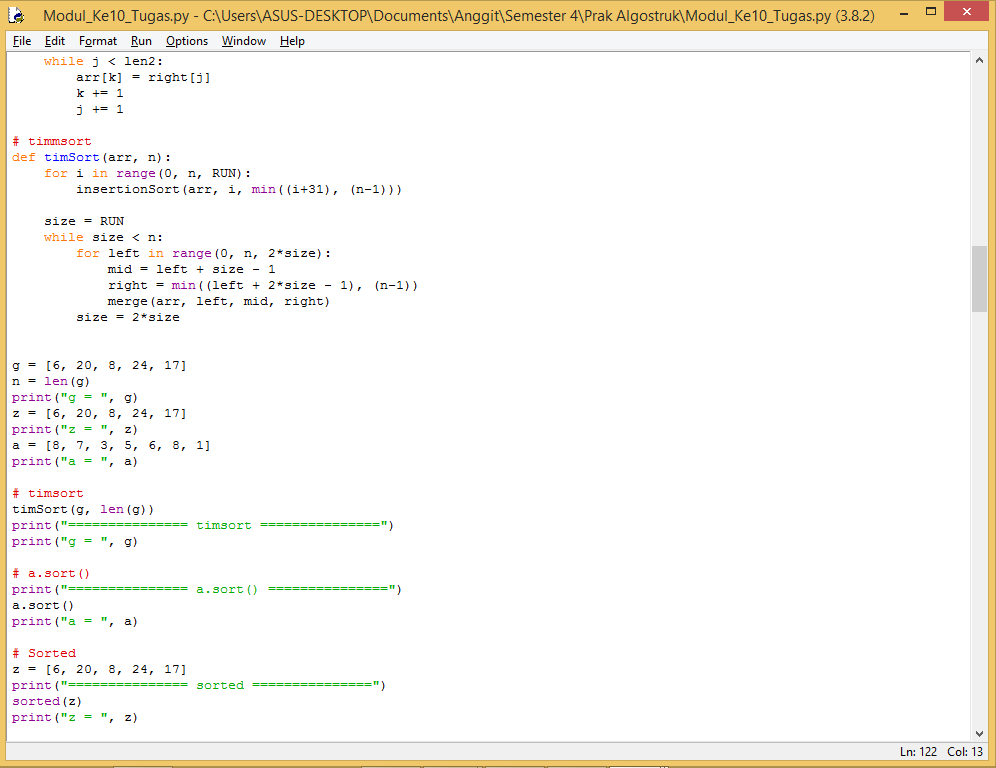


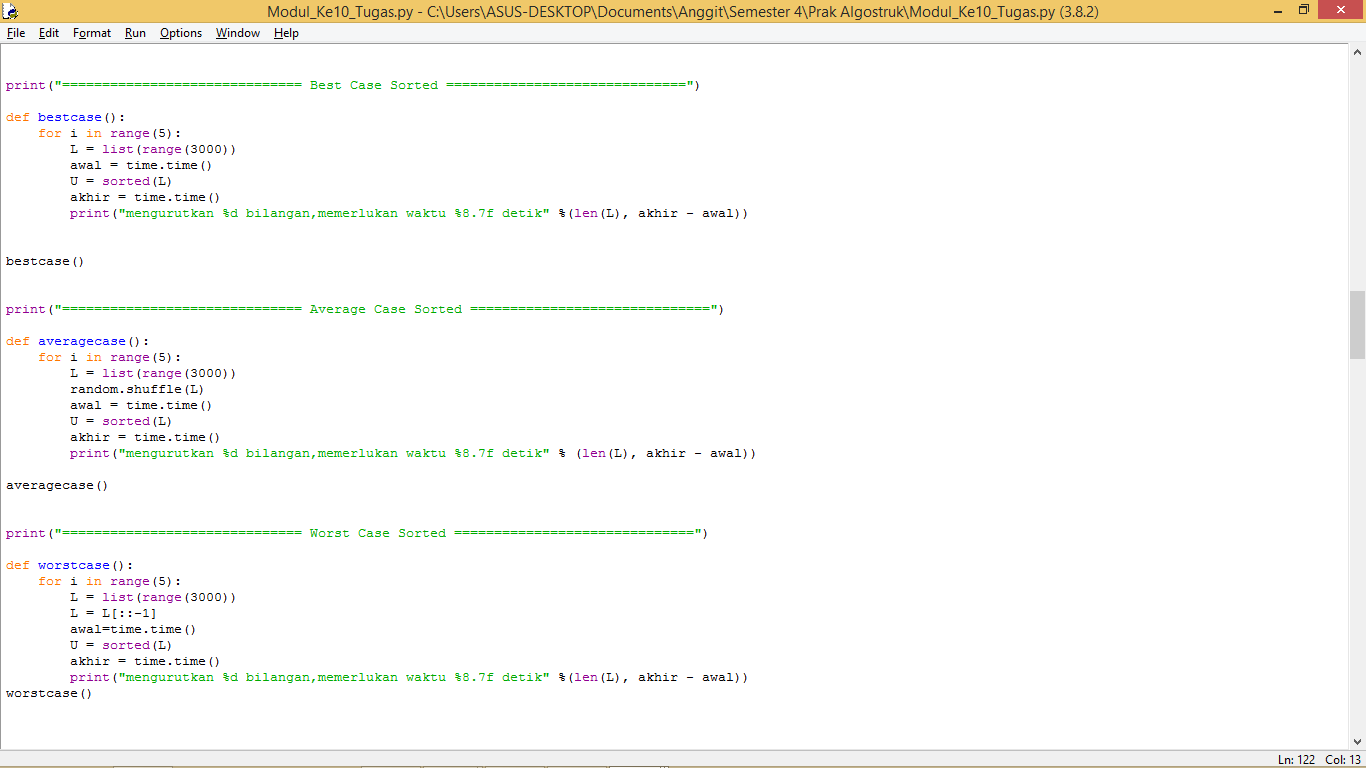
1. python mempunyai perintah untuk mengurutkan suatu list yang memanfaatkan algoritma Timesort. Jika g adalah suatu list berisi bilangan, maka g.sort() kan mengurutkannya. Perintah lain, sorted()mengurutkan list dan mengambalikan sebuah list baru yang sudah urut. Selidikilah fungsi sorted ini menggunakan timeit:

* apakah yang merupakan *best case* dan *average case* bagi sorted()?
* Confirm bahwa data input urutan terbaik bukan kasus terburuk bagi sorted().

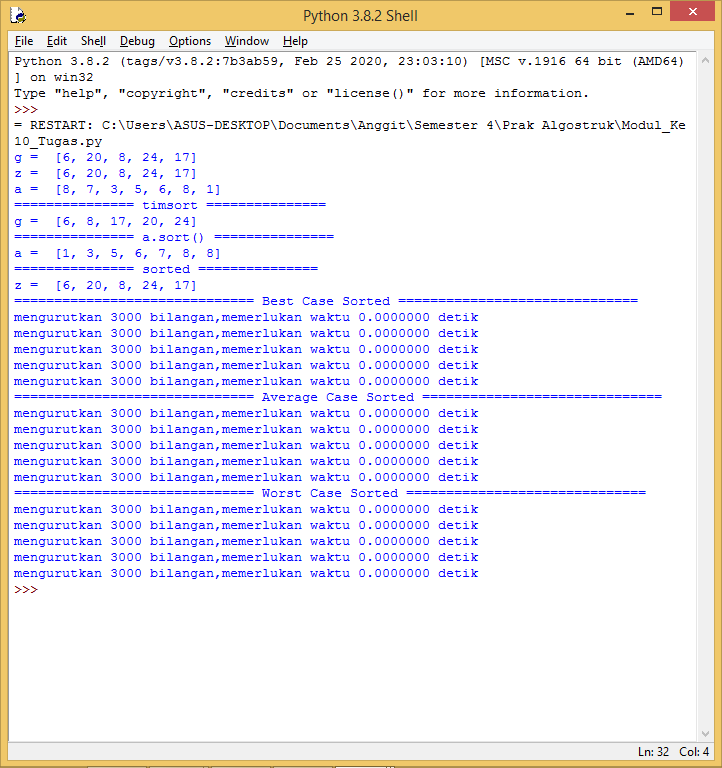
Source Code :



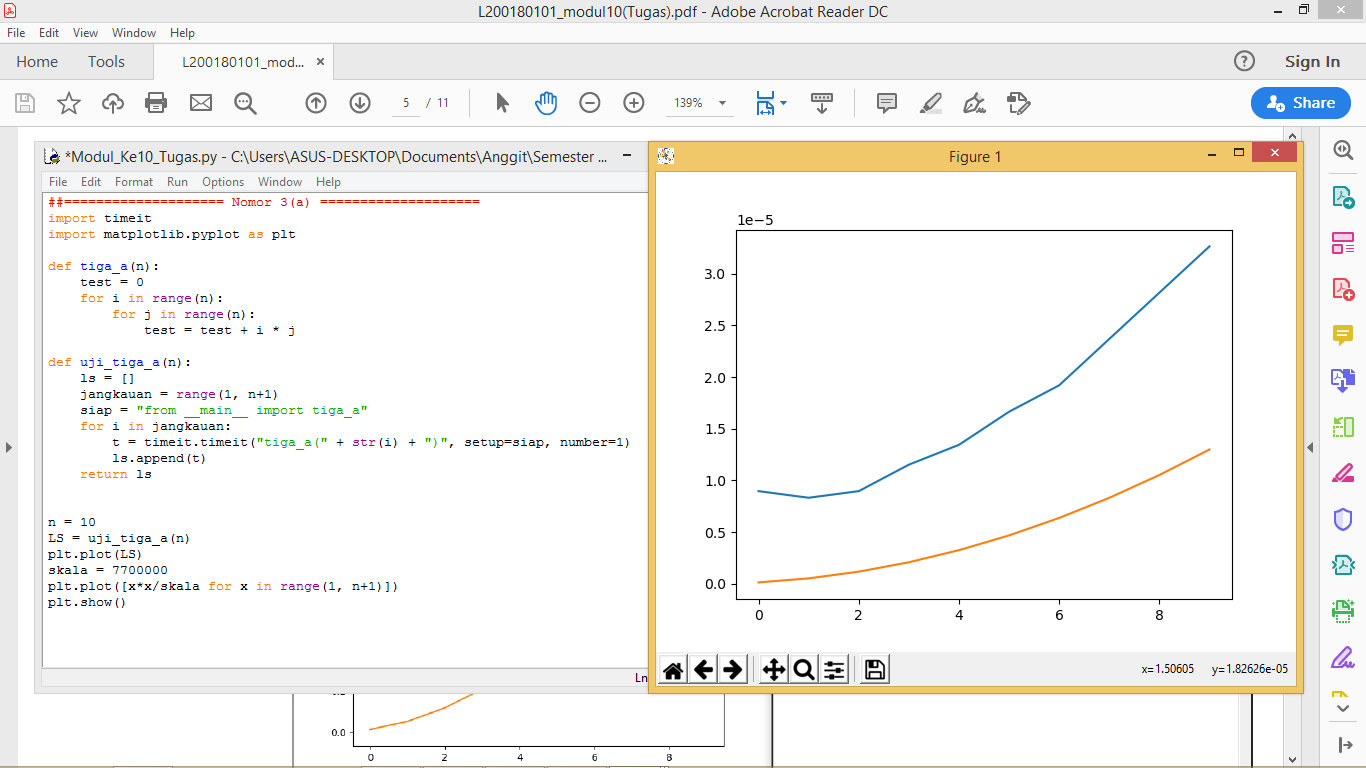




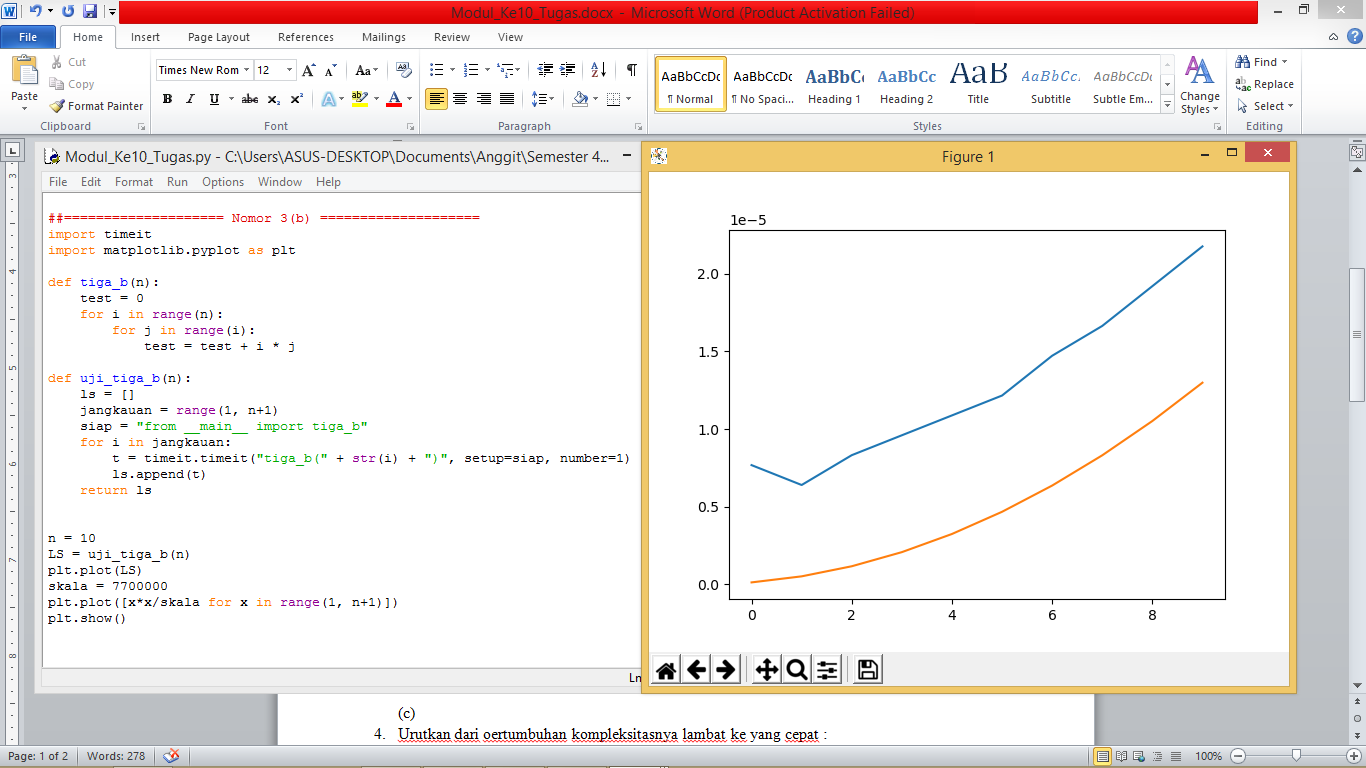
Hasil run :



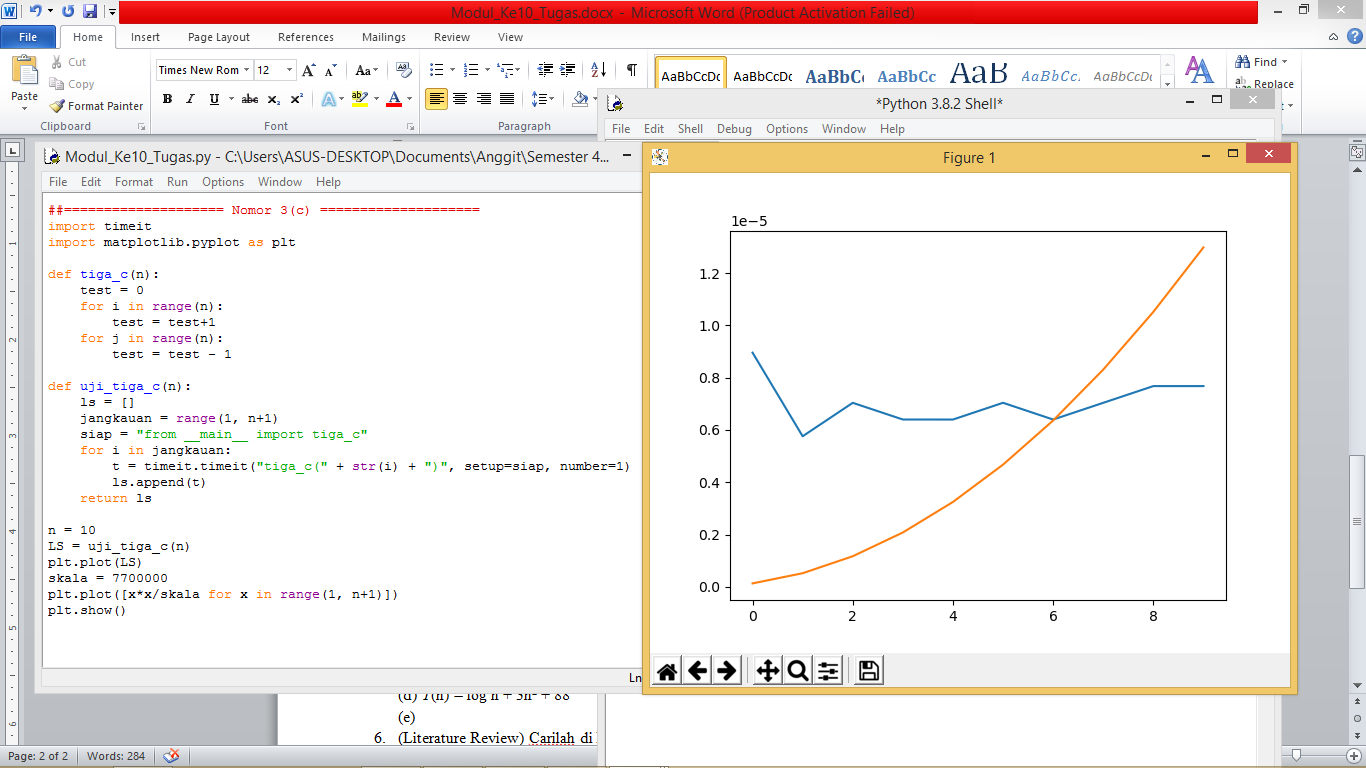
1. Untuk tiap kode berikut, tentukan *running time*-nya, O(1), O(log n), O(n), O(n log n), O(n2), atau O(n3), atau yang lain. Untuk memulai analisis, ambil suatu nilai n tertentu, lalu ikuti apa yang terjadi di kode itu.
2. Loop di dalam doop, keduanya sebanyak n : O(n2)



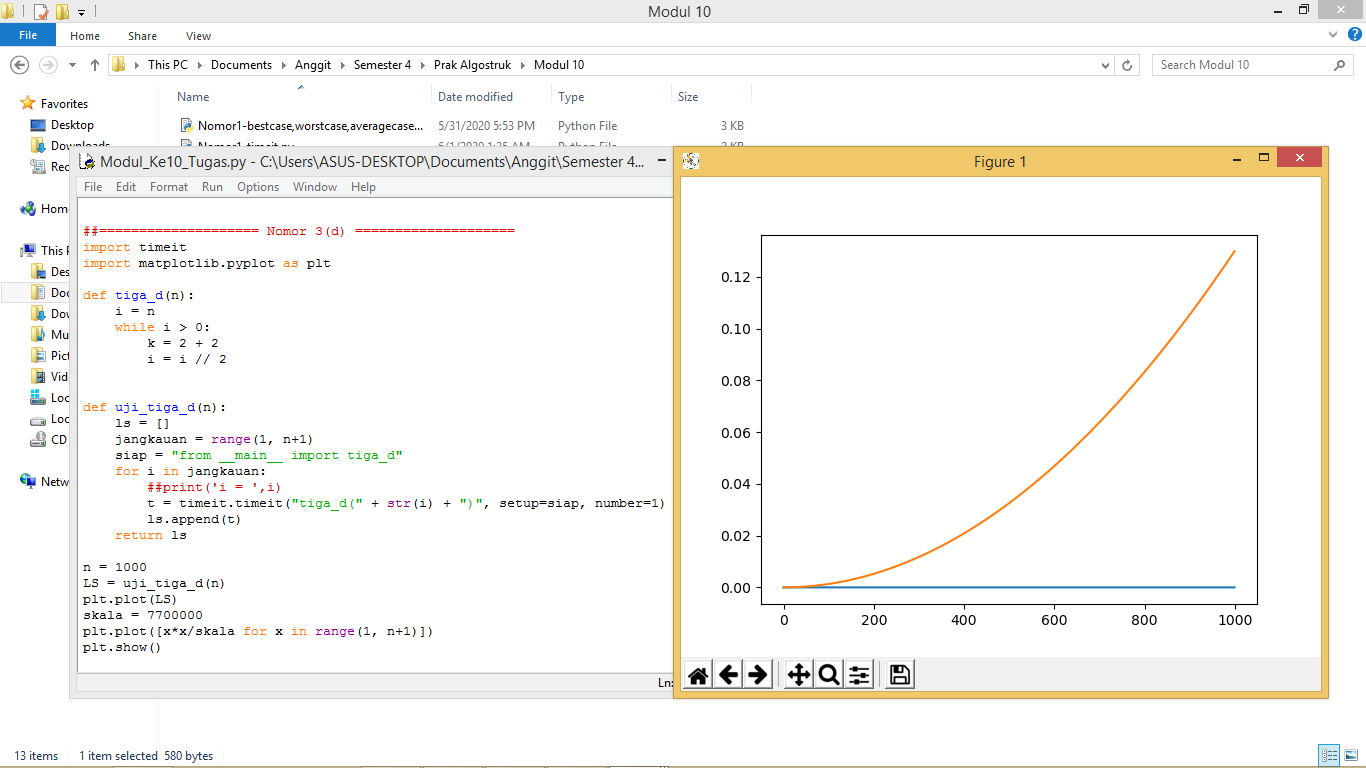
1. Loop di dalam loop, yang dalam bergantung nilai *i* loop luar : O(n log n)



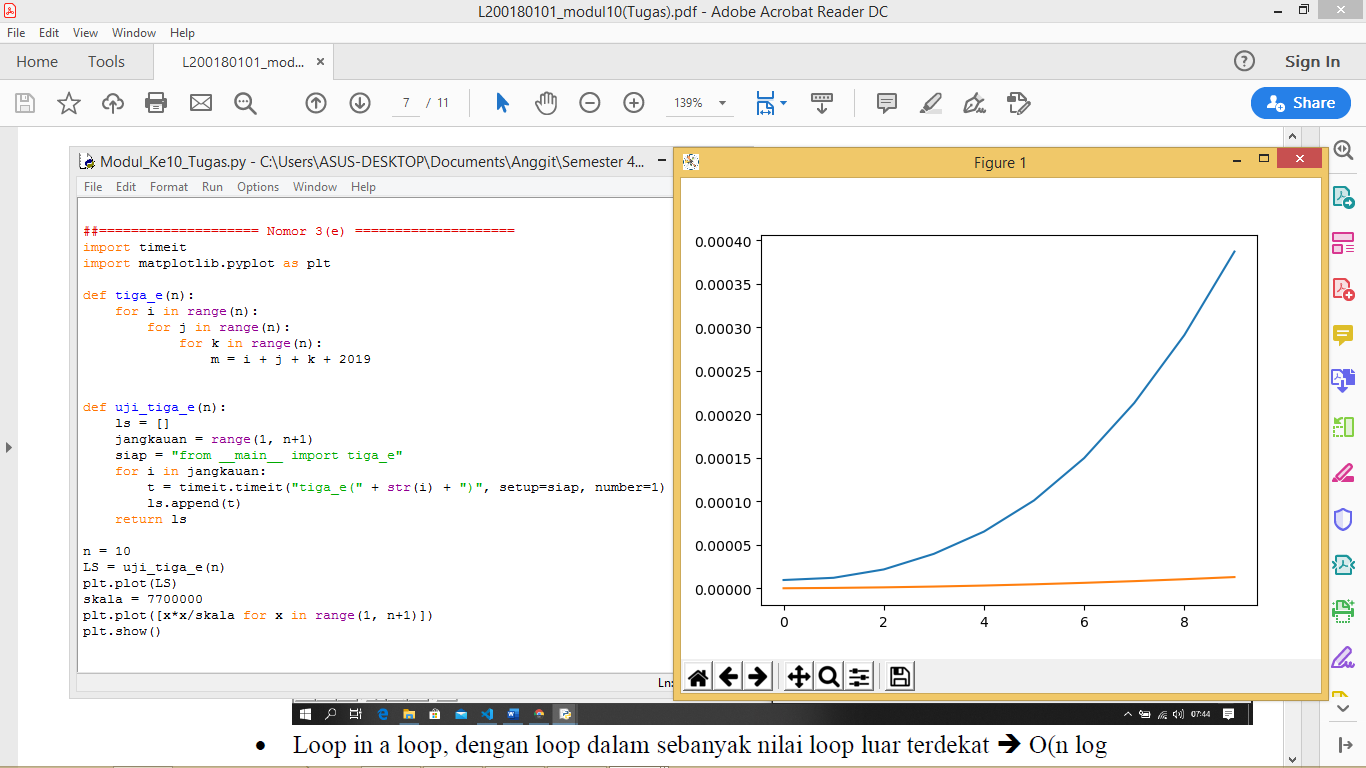
1. Dua loop terpisah : O(1)



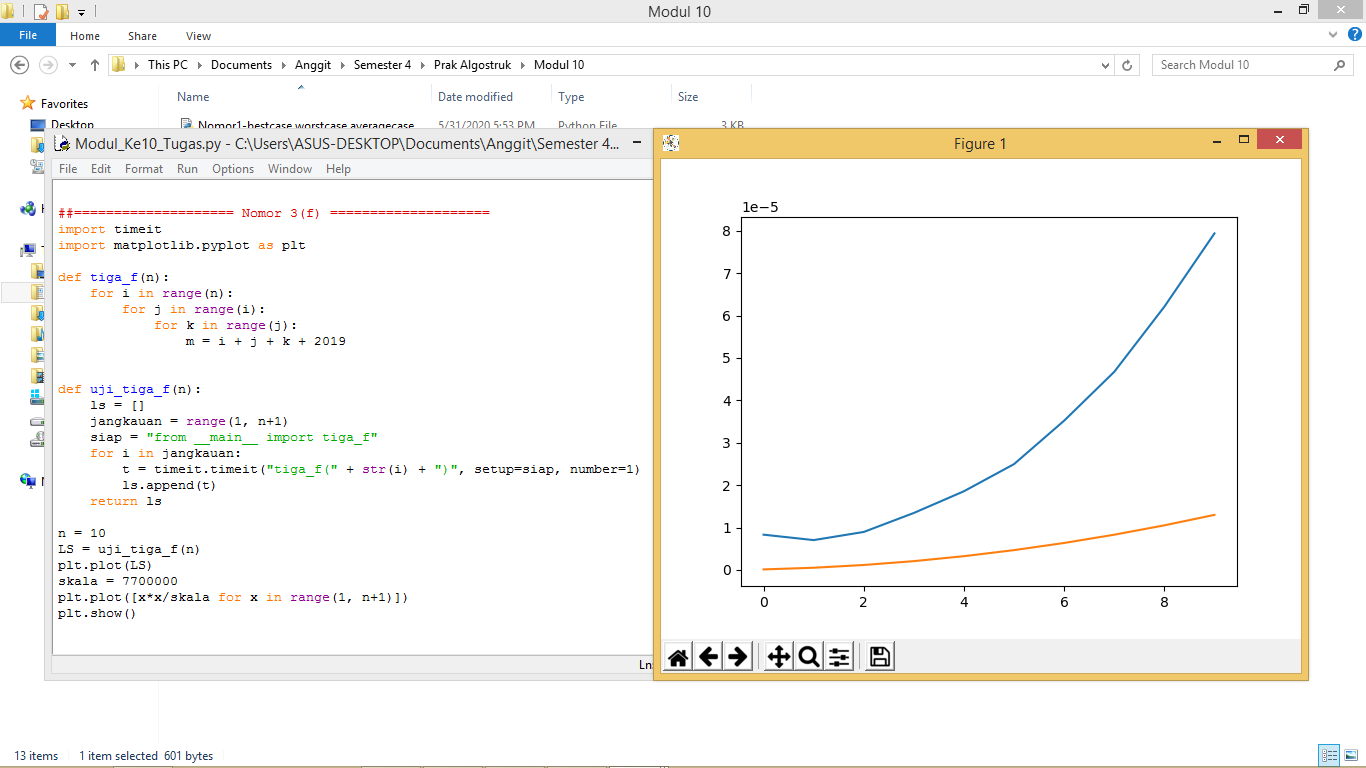
1. *While* loop yang dipangkas separuh tiap putaran : O(1)



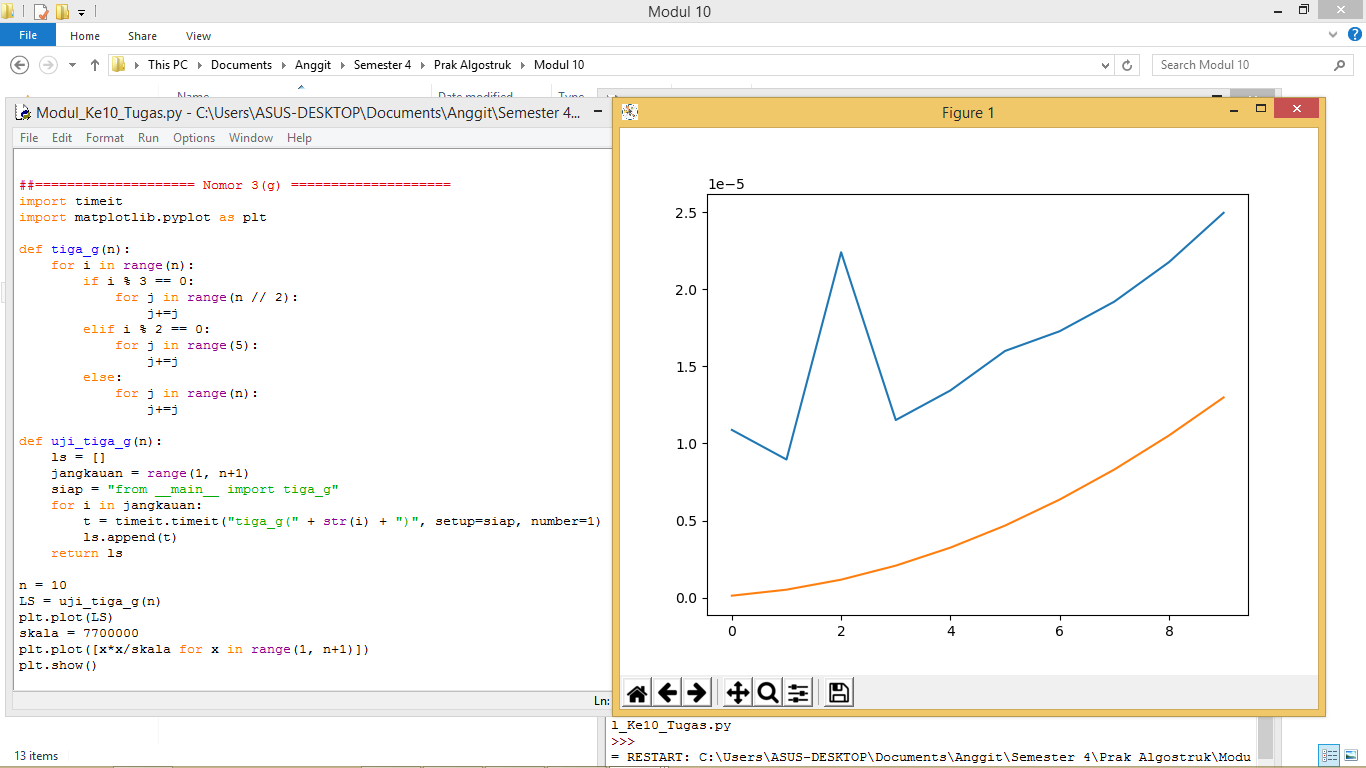
1. Loop in a loop in a loop, ketiganya sebanyak n : O(n3)



1. Loop in a loop in a loop, dengan loop dalam sebanyak nilai loop luar terdekat : O(n log n)



1. Fungsi ini : O(n)

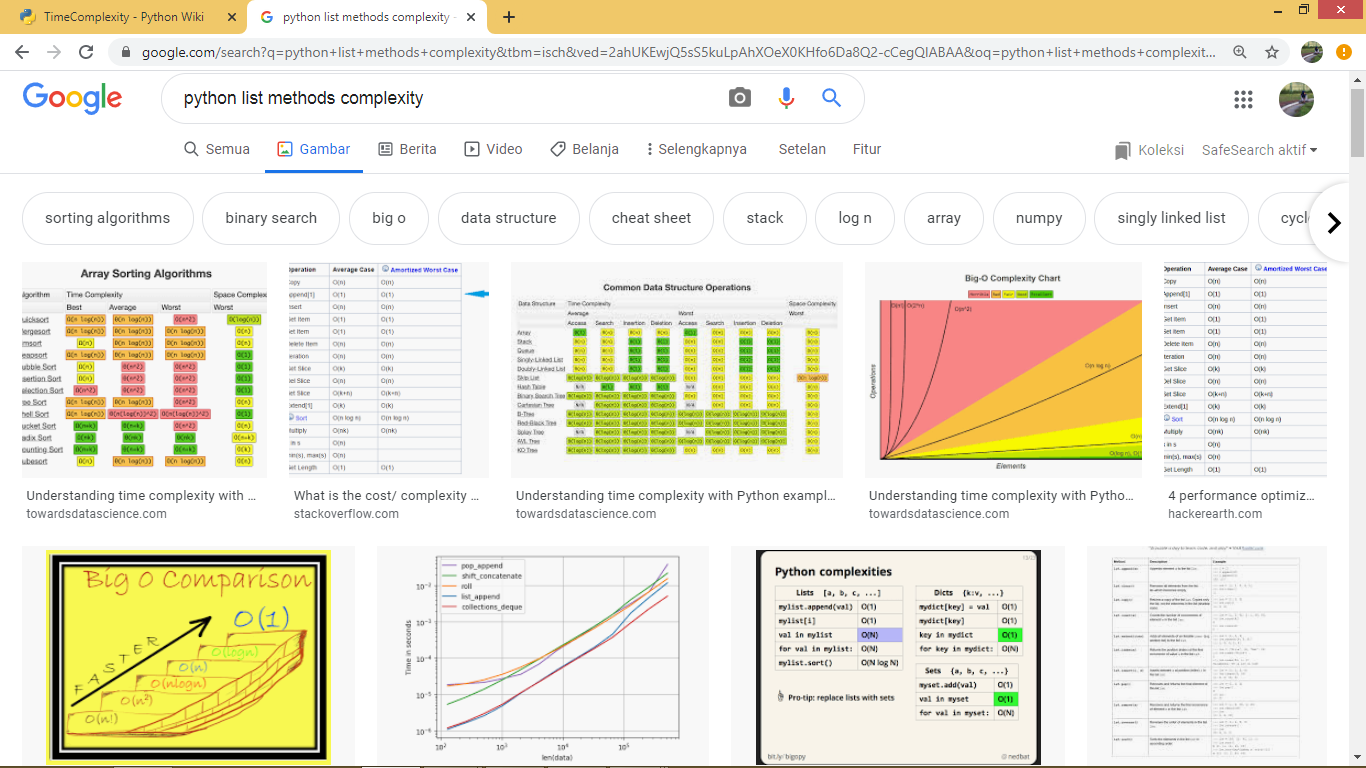


1. Urutkan dari pertumbuhan kompleksitasnya lambat ke yang cepat :

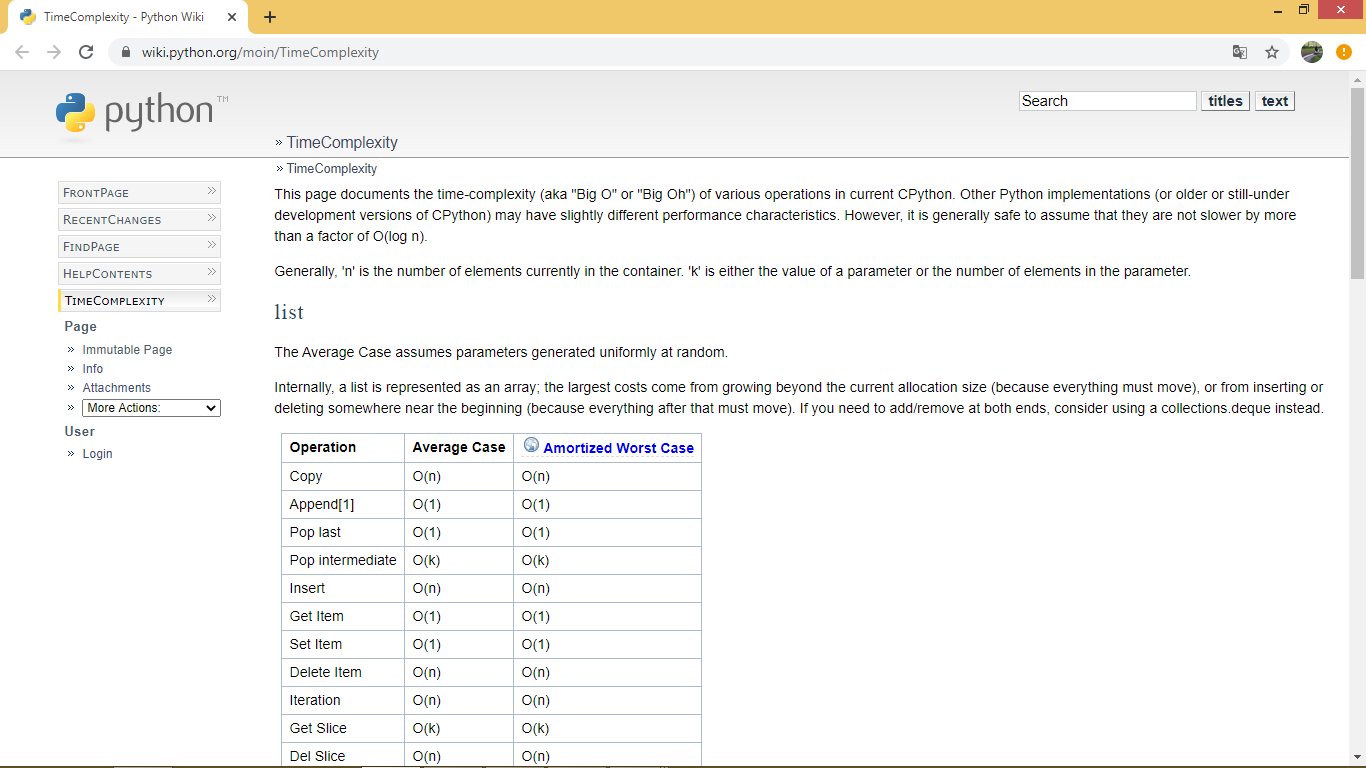
**n log2 n 4n 10 log2 n 5n2 log4 n 12n6 2log2n n3**

1. 10 log2 n
2. n log2 n
3. log4 n
4. 5n2
5. n3
6. 12n6
7. 2log2n
8. 4n
9. Tentukan O(.) dari fungsi-fungsi berikut, yang mewakili layaknya langkah yang diperlukan untuk beberapa algoritma.
10. *T*(n) = n2 + 32n + 8 → O(n2)
11. *T*(n) = 87n + 8n → O(n)
12. *T*(n) = 4n + 5n log n + 102 → O(n)
13. *T*(n) = log n + 3n2 + 88 → O(n2)
14. *T*(n) = 3(2n) + n2 + 547 → O(n2)
15. *T*(n,k) = kn + log (k) → O(kn)
16. *T*(n,k) = 8n + k log n + 800 → O(n)
17. *T*(n,k) = 100kn + n → O(nk)
18. (Literature Review) Carilah di Internet, kompleksitas metode\_metode pada object list di python. *Hint* :

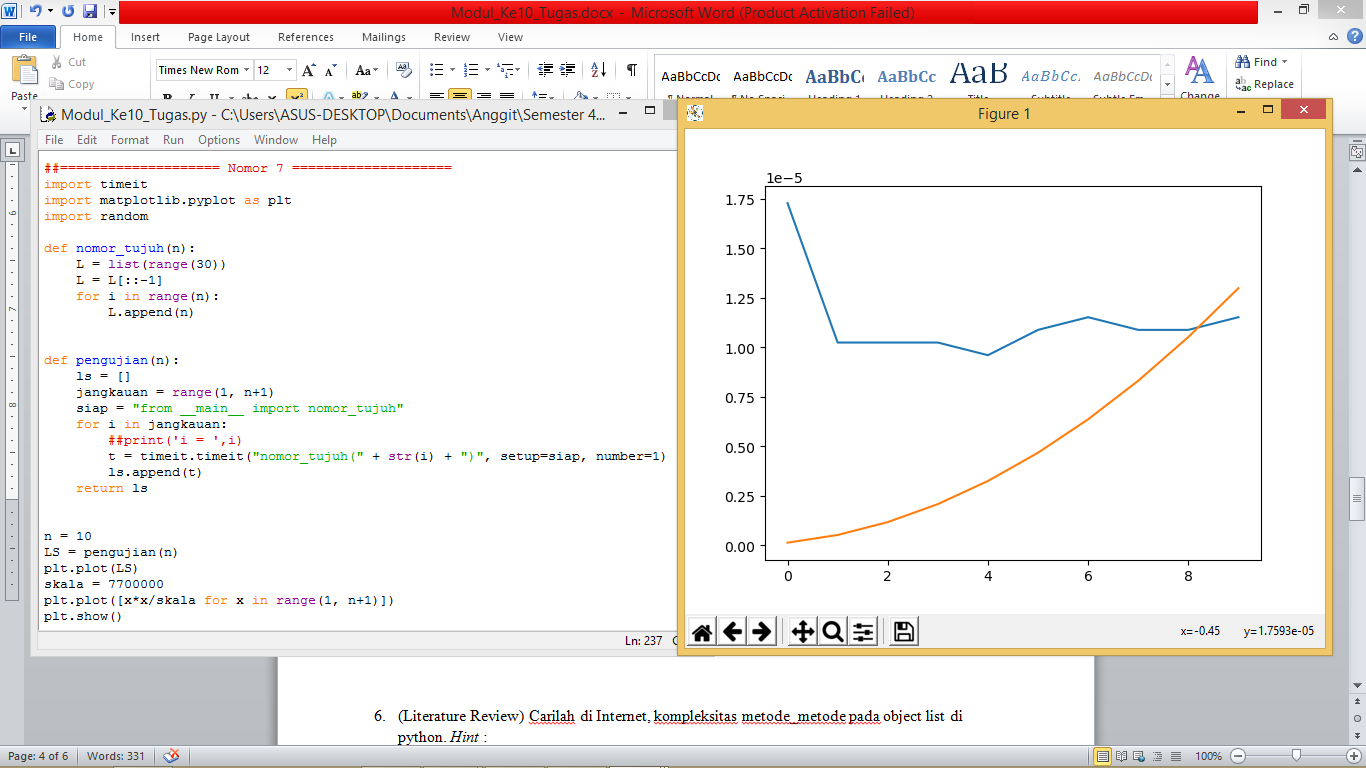
* Google python list methods complexity. Lihat juga bagian “Images”-nya



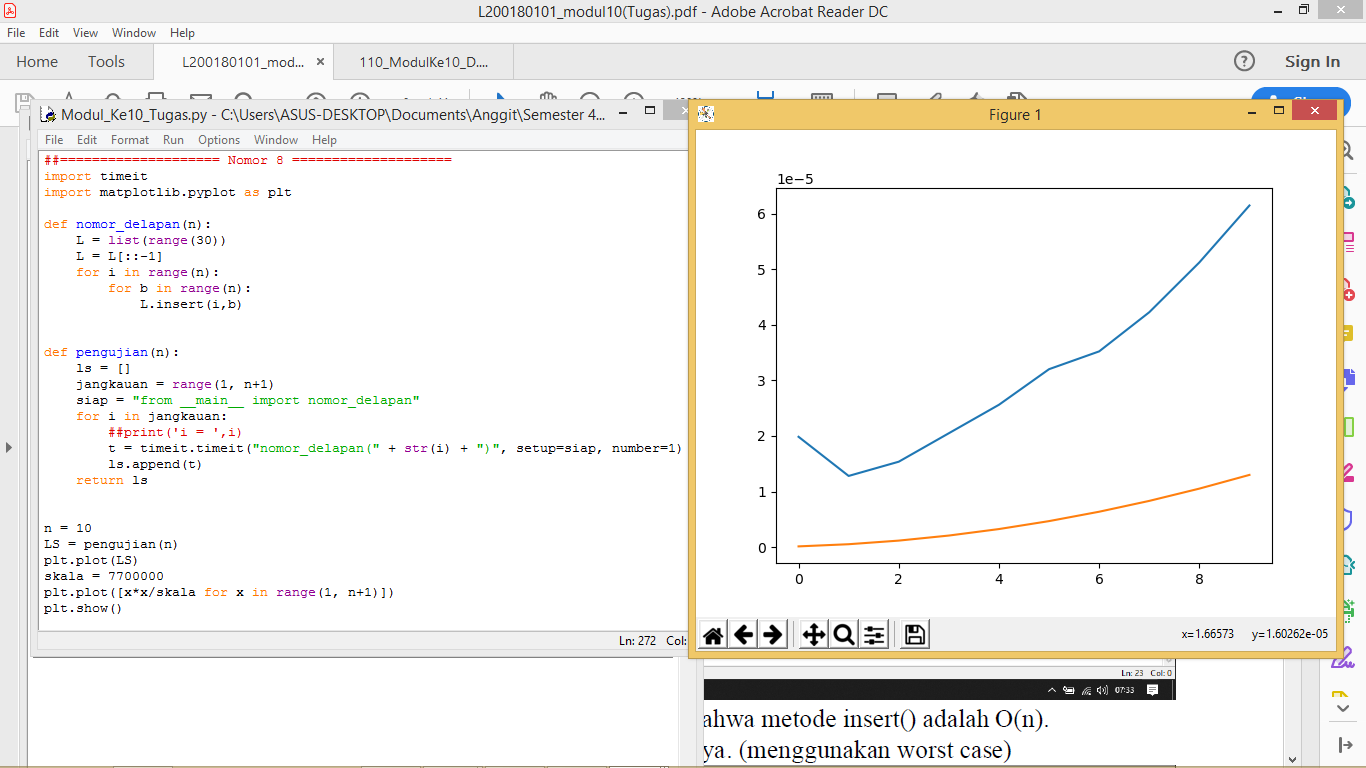
* Kunjungi <https://wiki.python.org/moin/TimeComplexity>



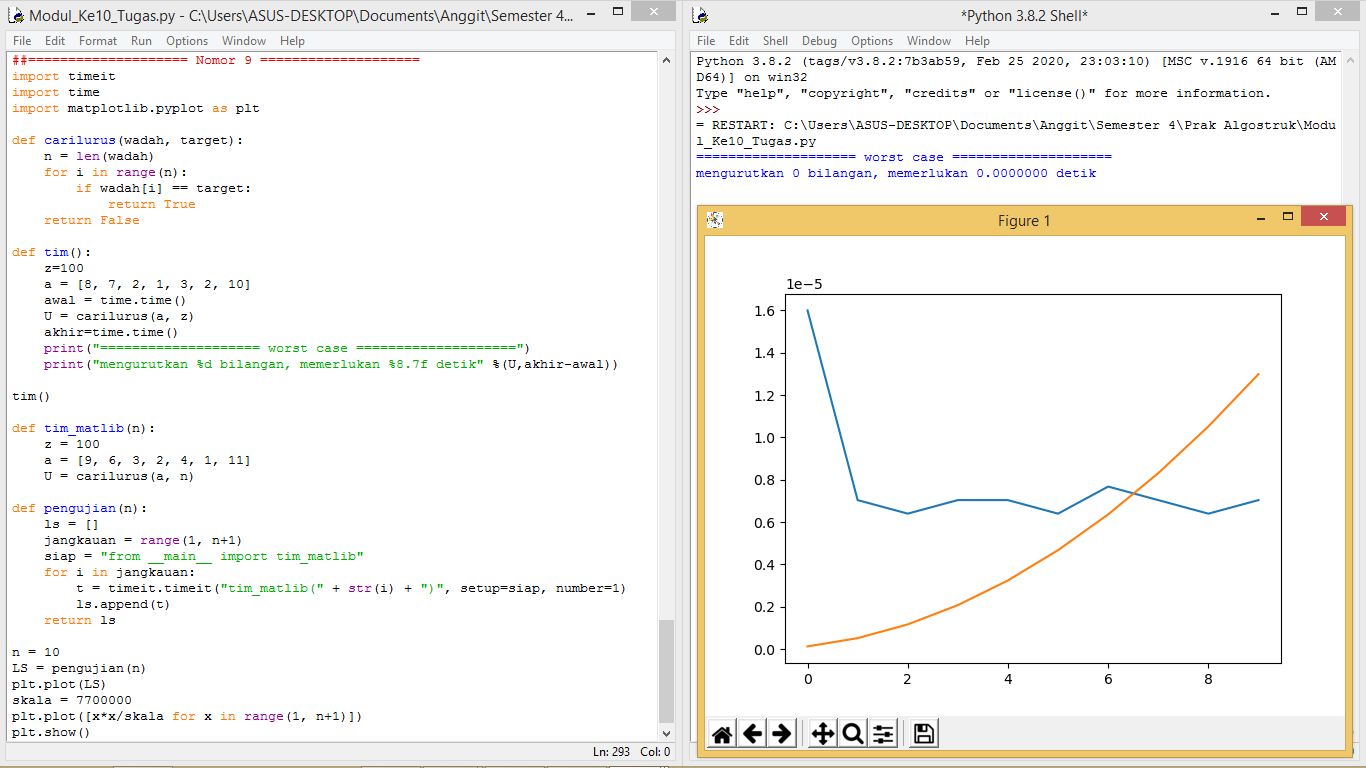
1. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode append() adalah O(1). Gunakan timeit dam matplotlib, seperti sebelumnya.



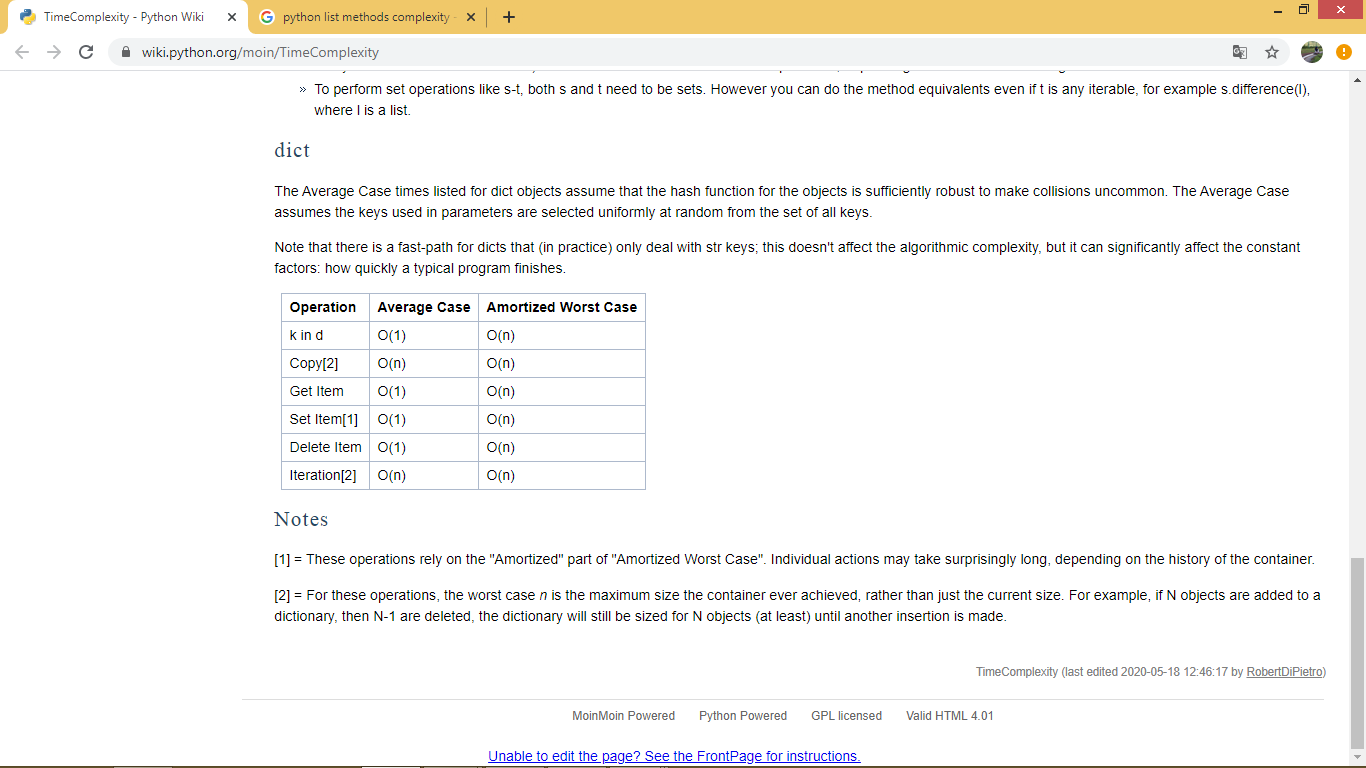
1. Buatlah suatu uji coba untuk mengkonfirmasi bahwa metode insert() adalah O(n). gunakan timeit dan matplotlib, seperti sebelumnya.



1. Buatlah suatu uji coba untuk mengkonfirmasi bahwa untuk memeriksa apakah-suatu-nilai-berada-di-suatu-list mempunyai kompleksitas O(n). gunakan timeit dan matplotlib seperti sebelumnya



1. (Literature review) Carilah di Internet, kompleksitas metode-metode pada object dict di python.



1. (Literature review) Selain notasi big-() O(.), ada pula notasi big-Theta Ѳ(.) dan notasi big-omega Ω(.) apakah beda diantara ketiganya ?

* **Big-O**

menggambarkan batas atas suatu algoritma.Dengan menggunakan contoh Penyisipan, laju pertumbuhan algoritma paling banyak kuadratik, atau O (n ^ 2).

f(n) = n2 − n < n2 = g(n).

* **Big-omega**

menjelaskan batas bawah suatu algoritma.Menggunakan contoh Penyisipan, jika input sudah diurutkan, maka laju pertumbuhan algoritma setidaknya linear, atau Ω (n).

f = Ω(g) if g = O(f)

* **Big-Theta**

menjelaskan batasan ketat dari suatu algoritma, batasnya dari atas dan bawah.Denganmenggunakan contoh Penyisipan akan diketahui tahu bahwa laju pertumbuhan paling banyak adalah O (n ^ 2) dan setidaknya Ω (n).

f = θ(g) if f = O(g) and f = Ω(g)

1. (Literaure review) Apa yang dimaksud dengan *amortized analysis* dalam analisis algoritma?

Amortized analysis adalah metode untuk menganalisis kompleksitas algoritma yang diberikan, atau berapa banyak sumber daya, terutama waktu atau memori, yang diperlukan untuk mengeksekusi. Amortized analysis menganalisa urutan operasi dan menjamin waktu rata-rata terburuk yang lebih rendah dari waktu dari operasi tertentu.