**LAPORAN**

**PRAKTIKUM ALGORITMA DAN STRUKTUR DATA**

**( MODUL 10 )**

**“ Analisis Algoritma”**

****

**Disusun oleh :**

**NAMA : CINDI DILA APRILIANA**

**NIM : L200200106**

**KELAS : E**

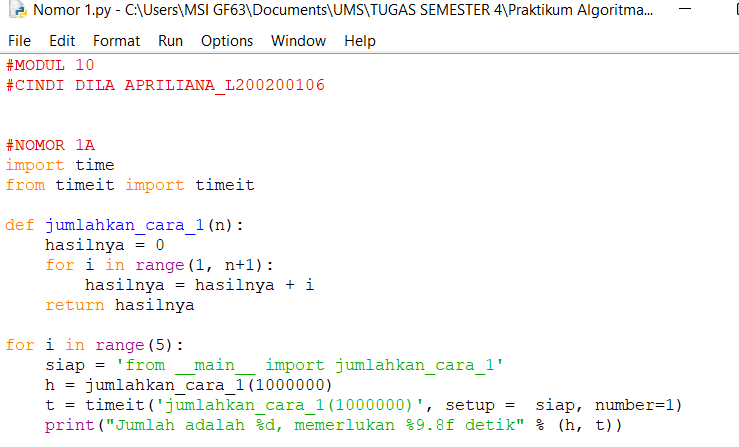
**INFORMATIKA**

**FAKULTAS KOMUNIKASI DAN INFORMATIKA**

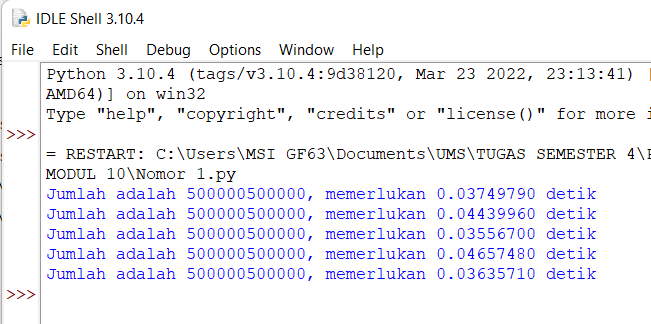
**UNIVERSITAS MUHAMMADIYAH SURAKARTA**

**TAHUN 2021/2022**

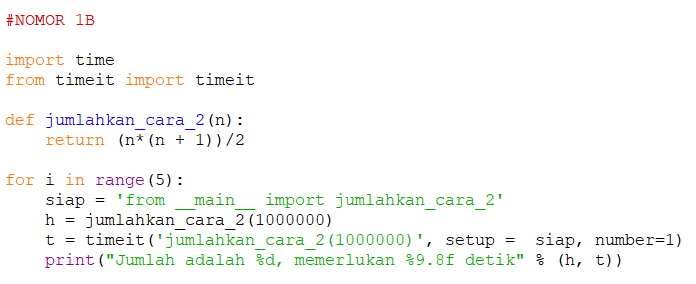
1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul timeit, yakni
2. Jumlahkan\_cara\_1

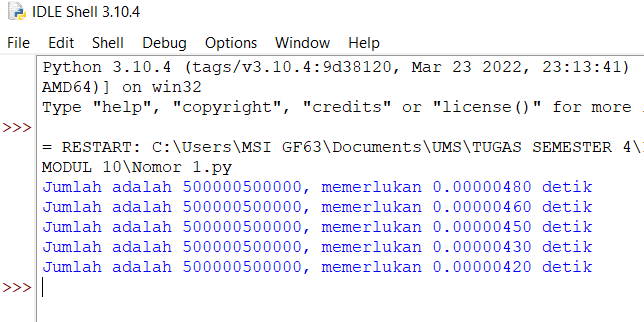


Output

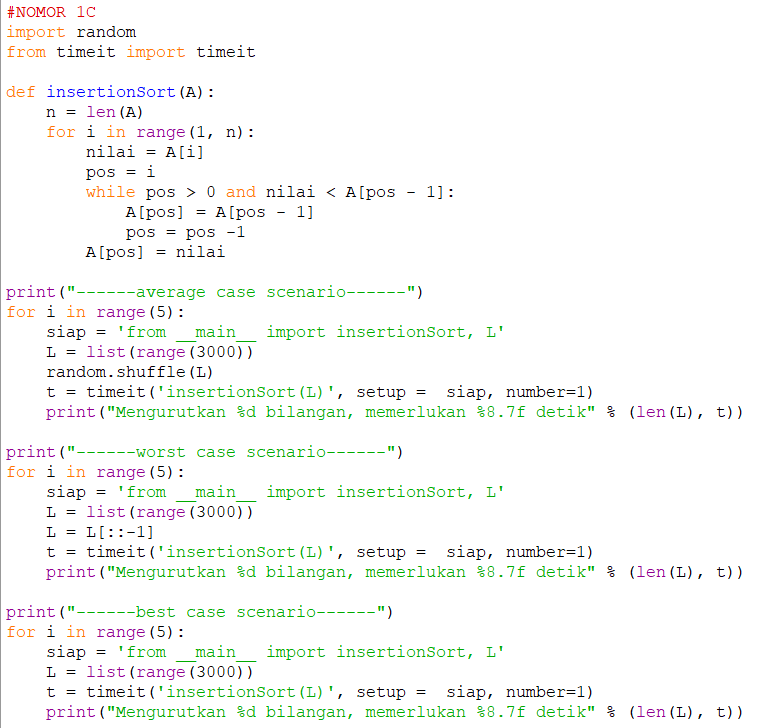


1. jumlahkan\_cara\_2

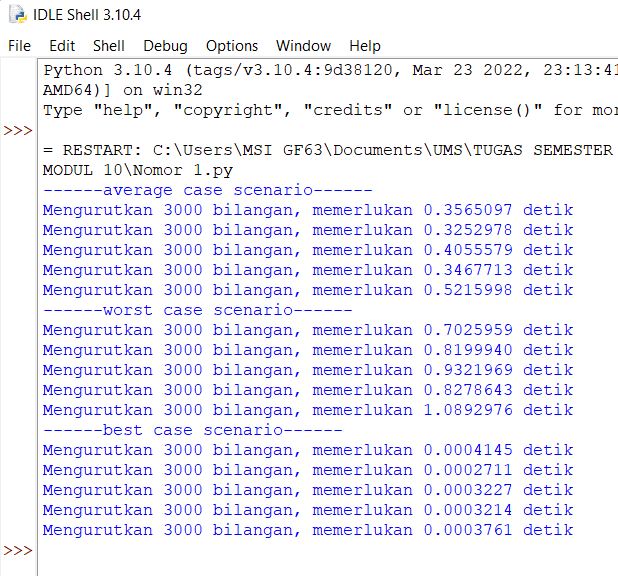


Output 

1. insertionSort



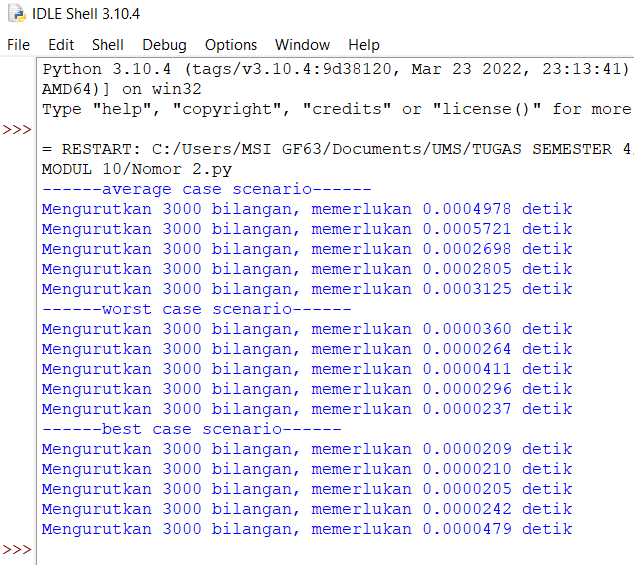
Output



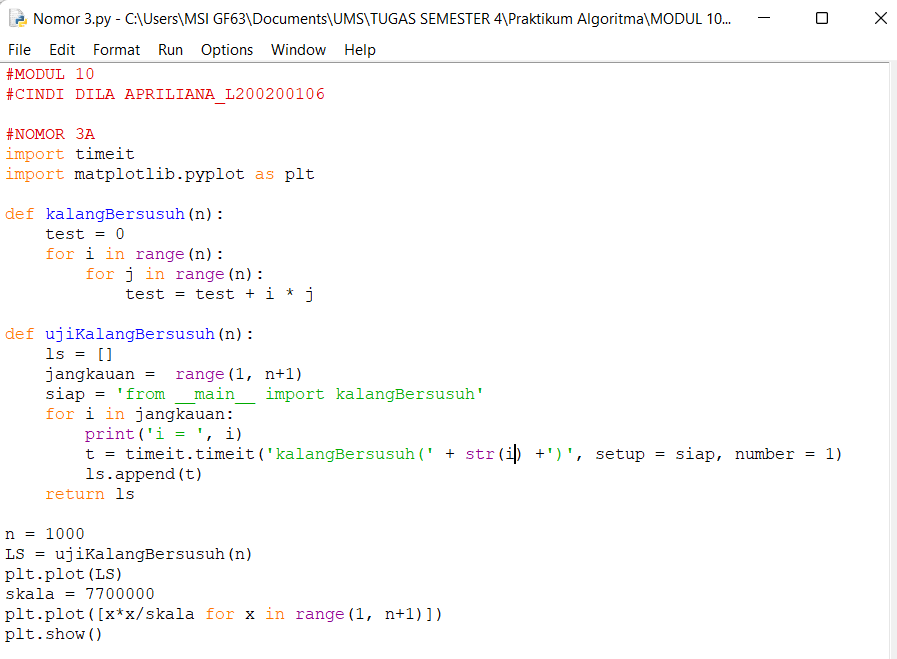
1. Algoritma timsort



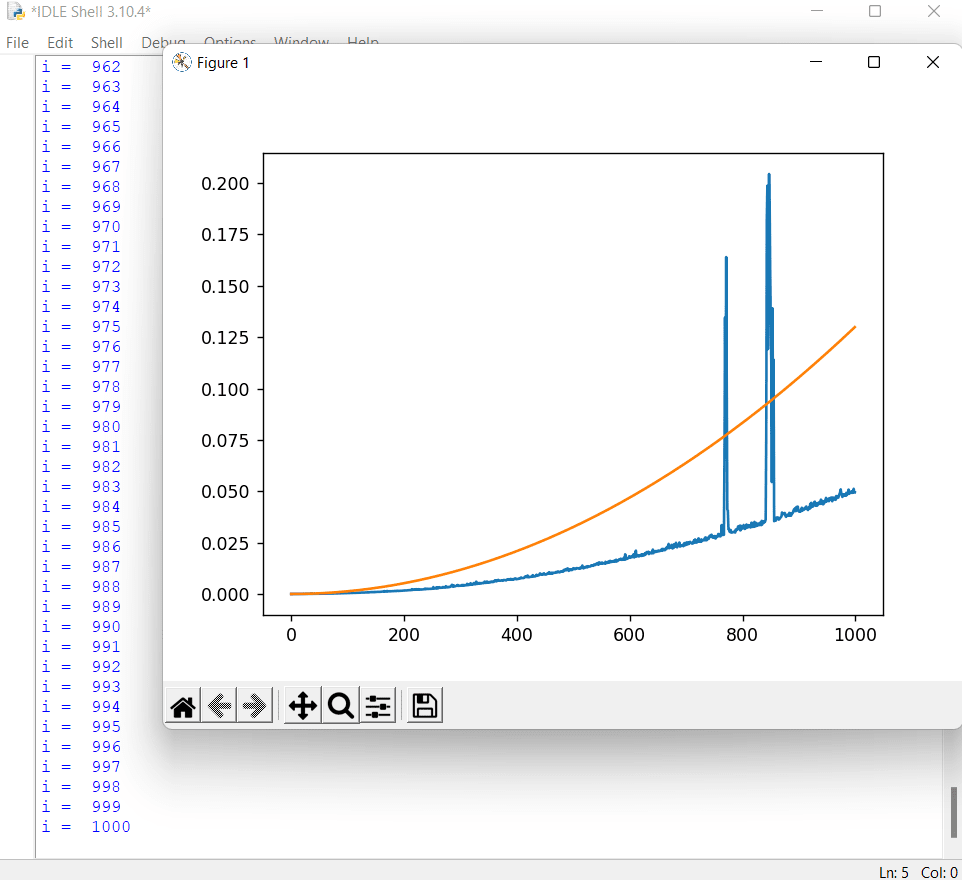
Output



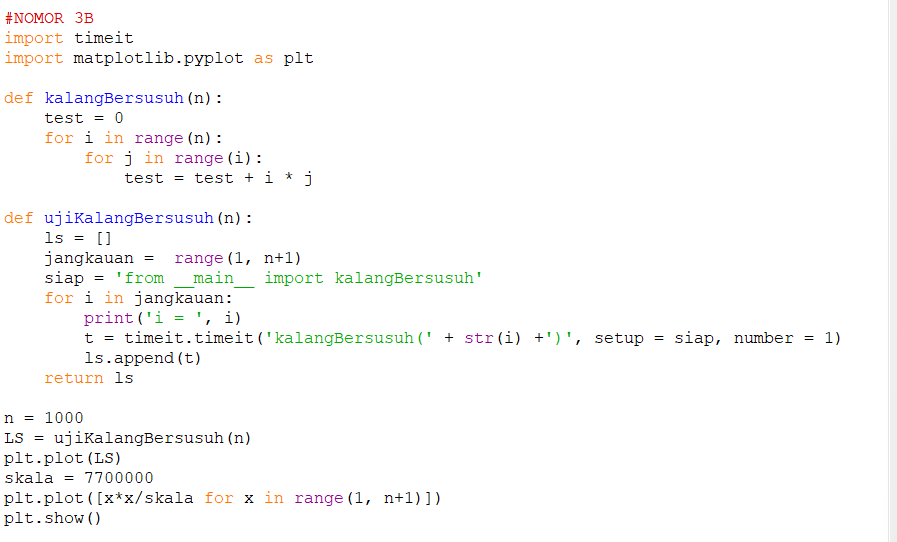
1. Tentukan running time-nya, O(1), O(log n), O(n), O(n log n), O(n2 ), atau O(n3)
2. Loop di dalam loop, keduanya sebanyak



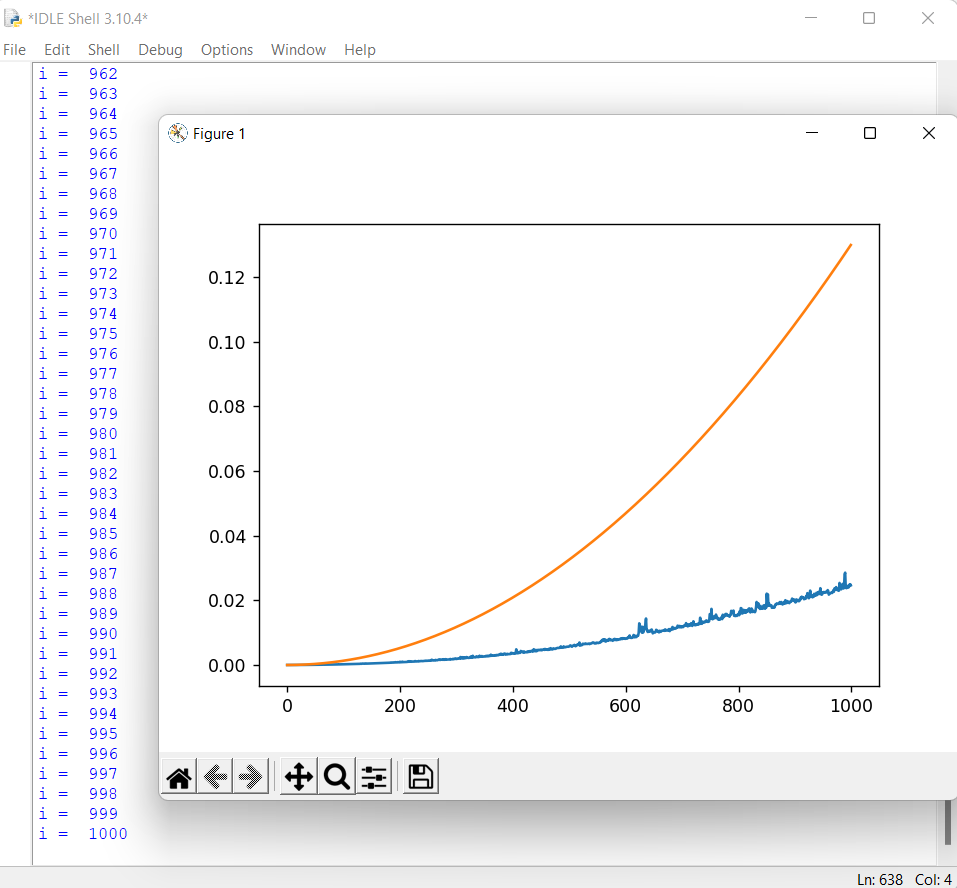
Output



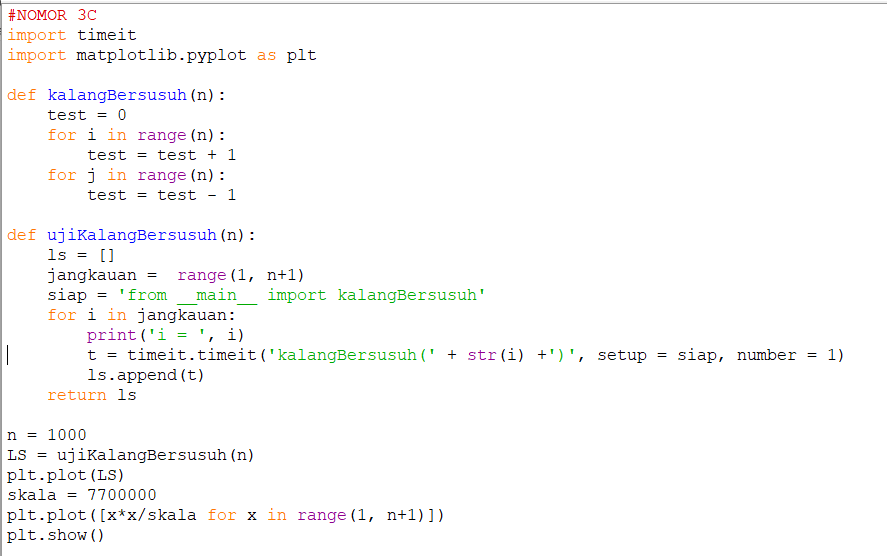
1. Loop di dalam loop, yang dalam bergantung n



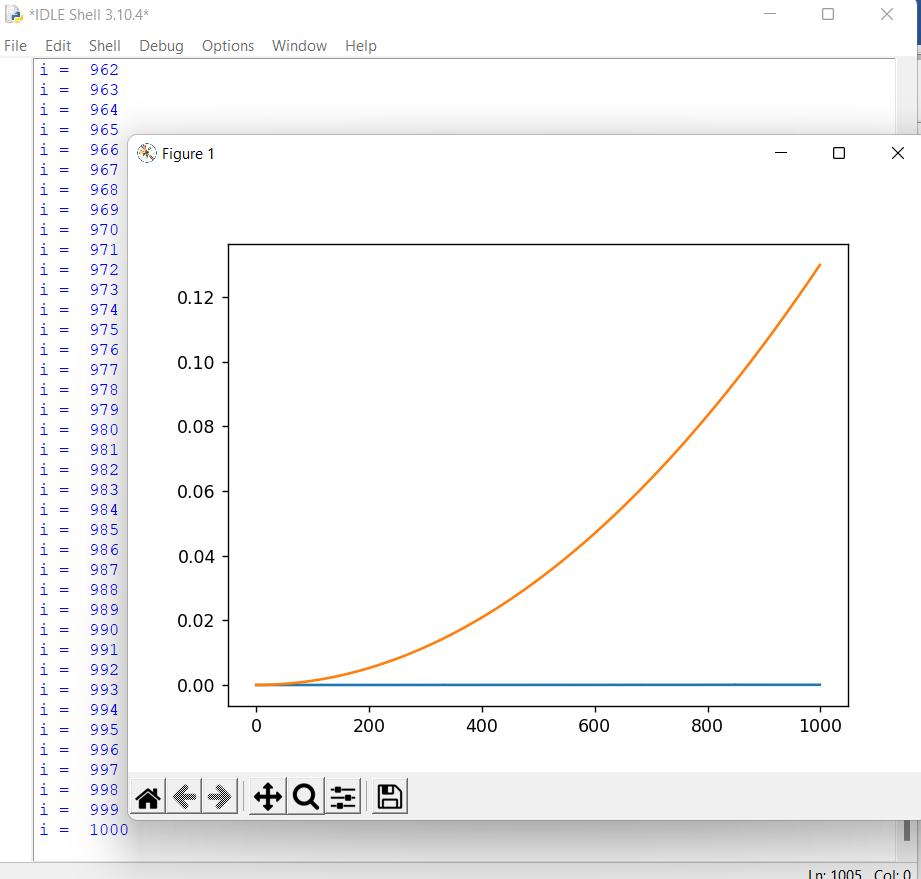
Output



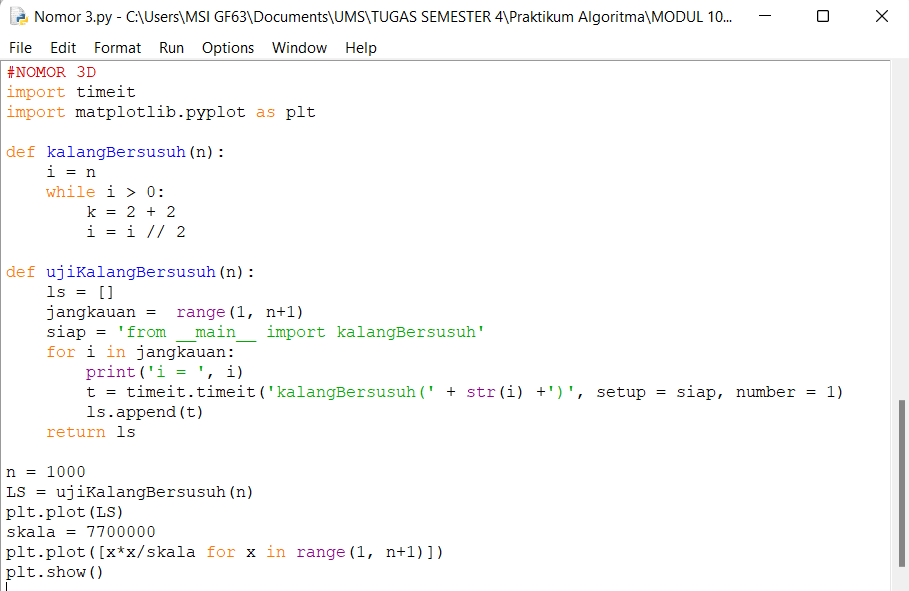
1. Dua loo



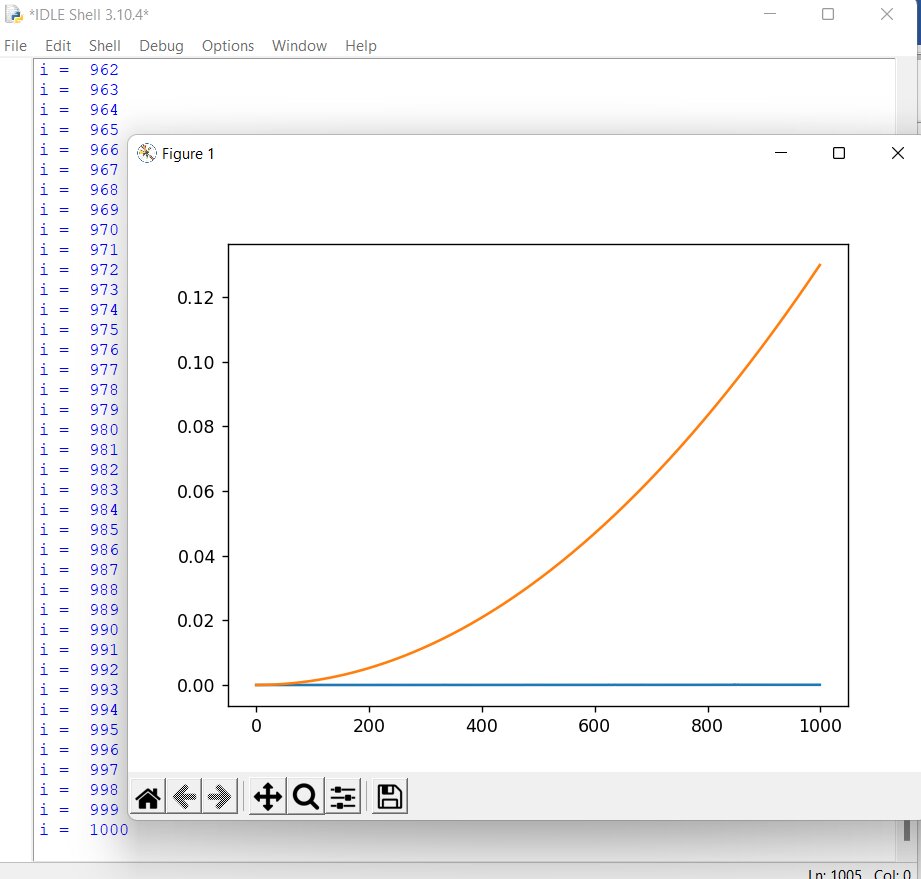
Output



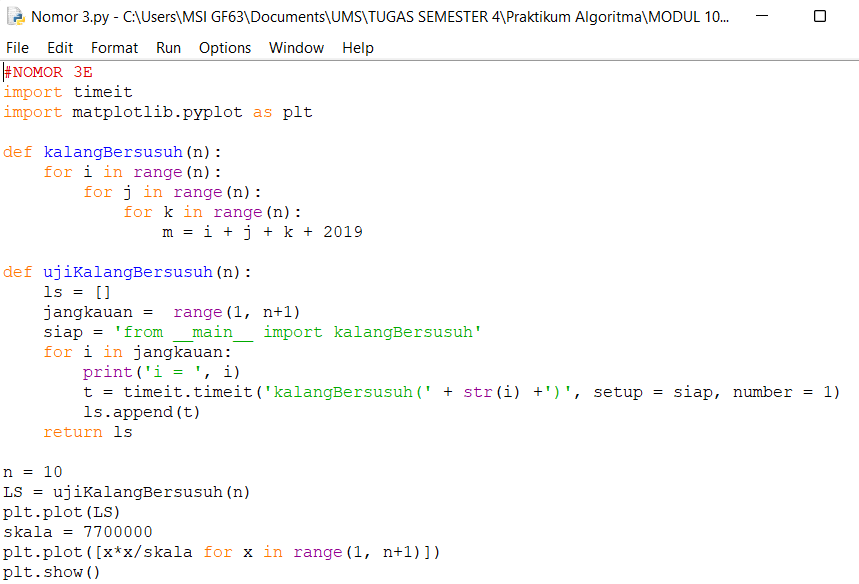
1. While loop yang dipangkas separuh tiap putaran



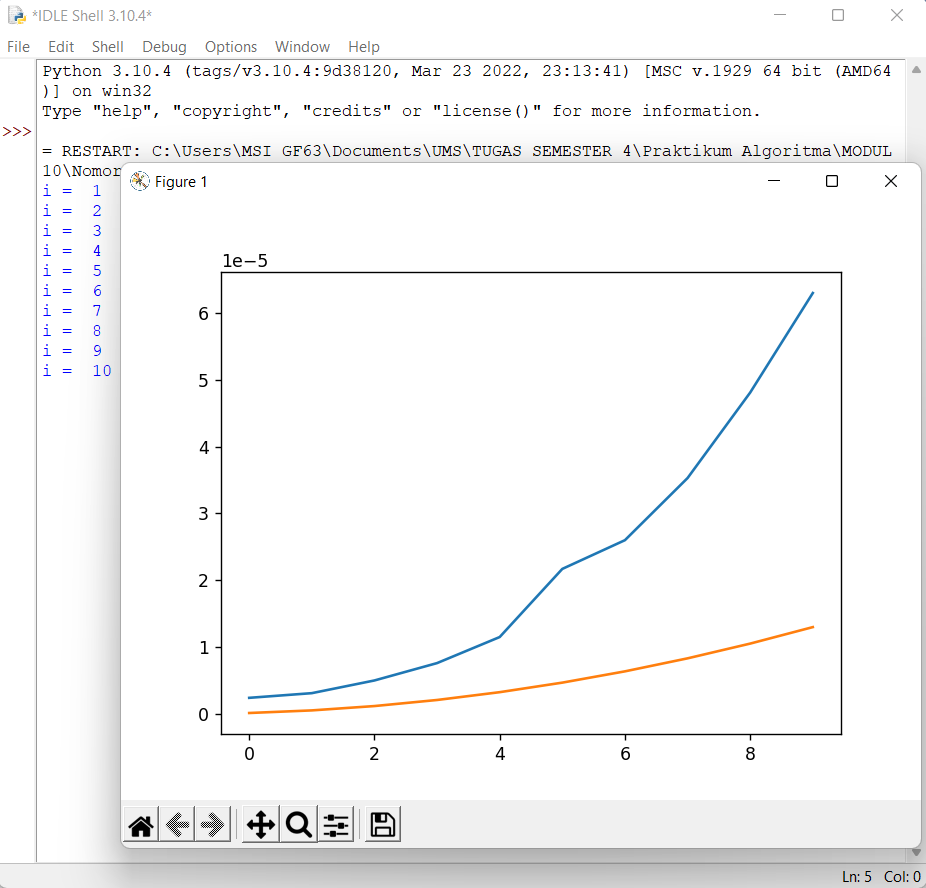
Output

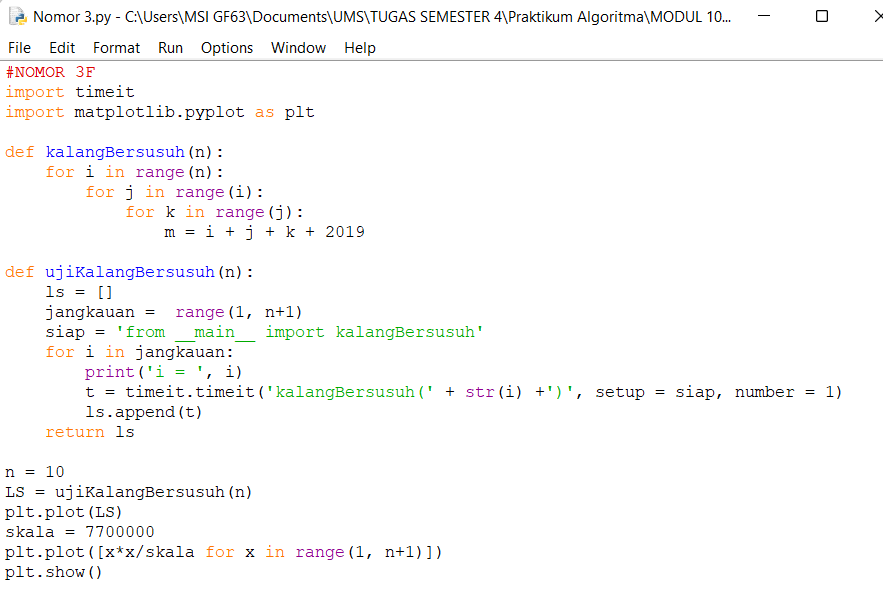


1. Loop di dalam loop, ketiganya sebanyak n

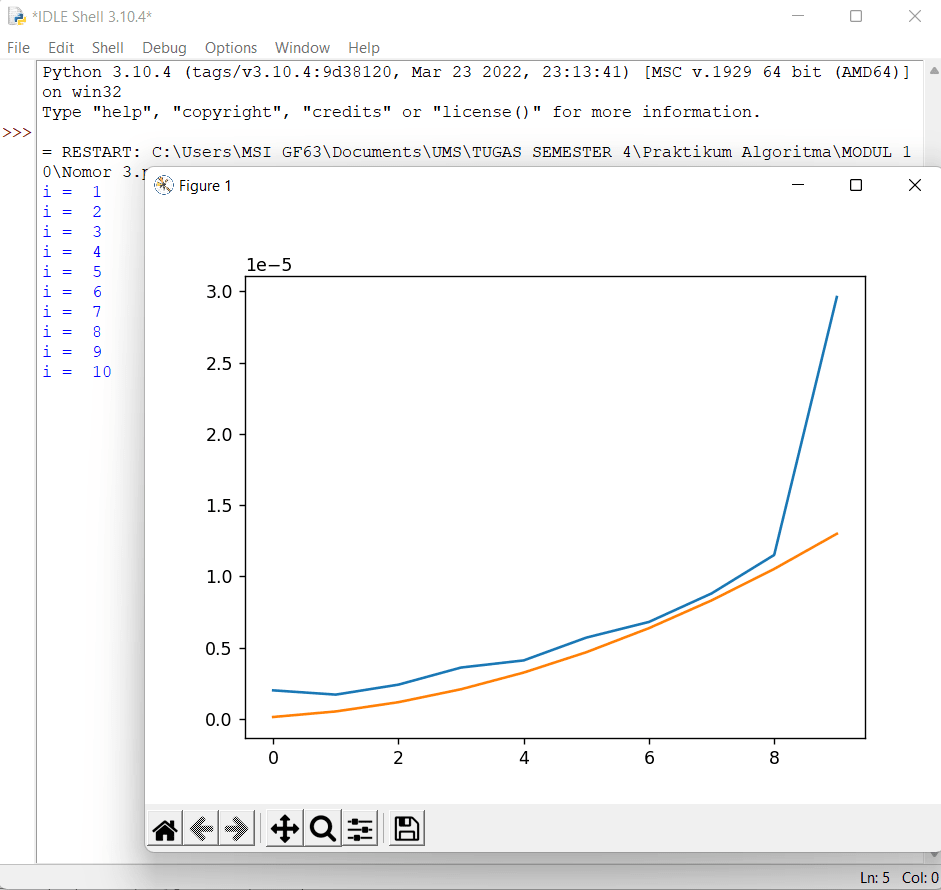


Output



1. Loop di dalam loop, dengan loop dalam sebanyak nilai loop luar terdekat

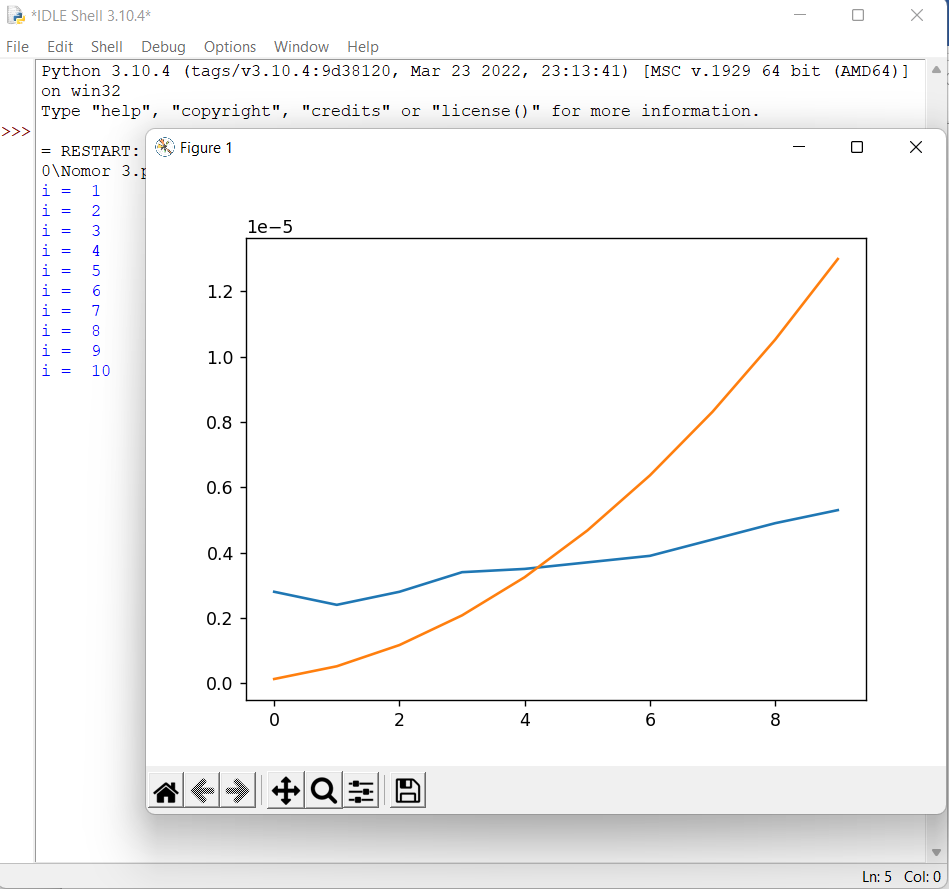
Output



1. Fungsi i



Output



1. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat

log4*n* < 10log2*n* < *n* log2*n* < 2 log2*n* < 5*n*2 < *n*3 < 12*n*6 < 4*n*

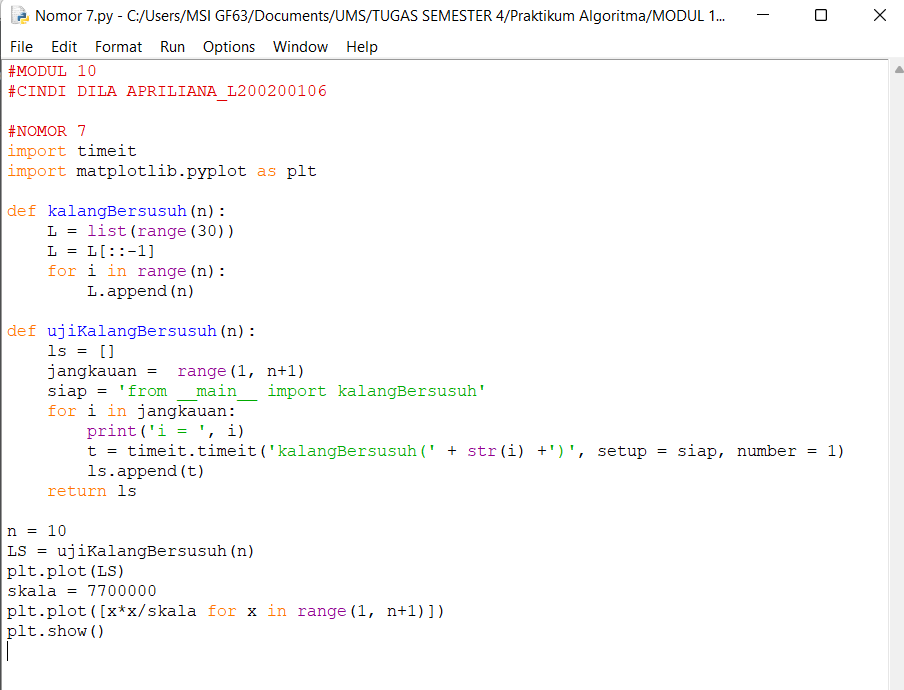
1. Tentukan O(.) dari fungsi-fungsi berikut, yang mewakili banyaknya Langkah yang diperlukan untuk beberapa algoritma
   1. T(*n*) = *n*2 + 32*n* + 8 = O(*n*2)
   2. T(*n*) = 87*n* + 8*n* = O(*n*)
   3. T(*n*) = 4*n* + 5*n* log*n* + 102 = O(*n* log*n*)
   4. T(*n*) = log*n* + 3*n*2 + 88 = O(*n*2)
   5. T(*n*) = 3(2*n*) + *n*2 + 647 = O(2*n*)
   6. T(*n*, *k*) = *kn* + log*k* = O(*kn*)
   7. T(*n*, *k*) = 8*n* + *k* log*n* + 800 = O(*n*)
   8. T(*n*, *k*) = 100*kn* + *n* = O(*kn*)
2. Carilah di internet, kompleksitas metode-metode pada object list di Python. Hint

* Google ‘Python list methods complexity’. Lihat juga bagian ‘images’
* Kunjungi <https://wiki.python.org/moin/TimeComplexity>

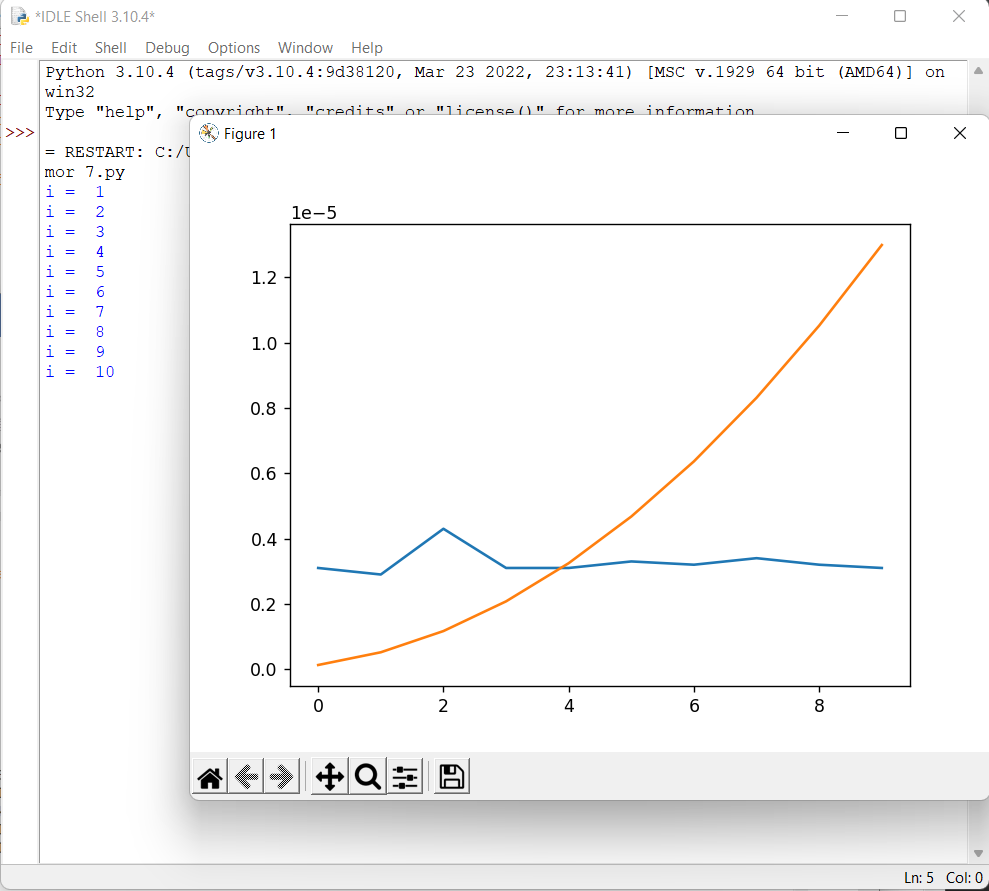
Kasus Rata-rata mengasumsikan parameter yang dihasilkan seragam secara acak.

Secara internal, list direpresentasikan sebagai array; biaya terbesar berasal dari pertumbuhan di luar ukuran alokasi saat ini (karena semuanya harus bergerak), atau dari memasukkan atau menghapus suatu tempat di dekat awal (karena semuanya setelah itu harus bergerak). Jika perlu kita menambahkan/menghapus di kedua ujungnya, pertimbangkan untuk menggunakan collections.deque sebagai gantinya.

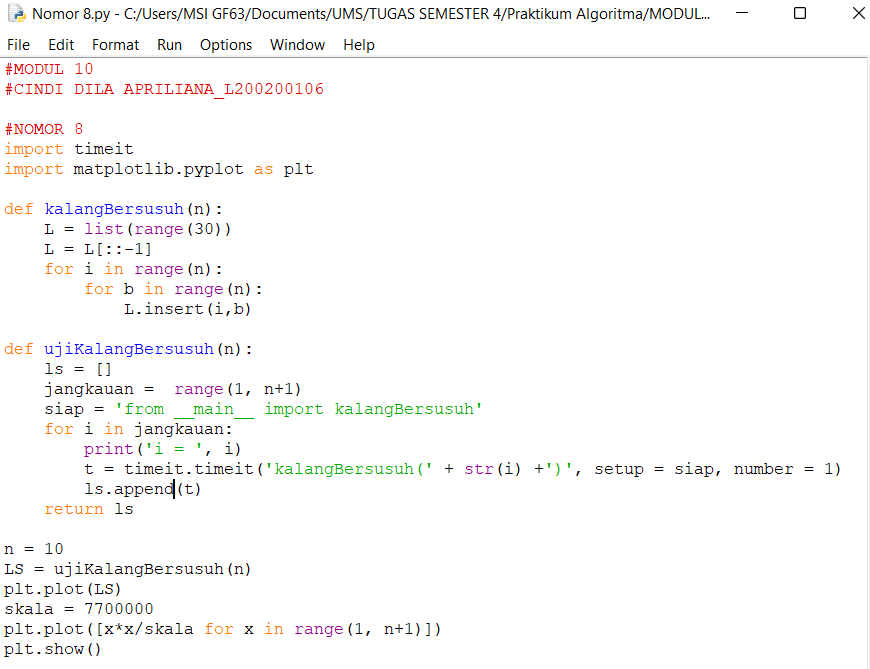
1. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode append() adalah O(1).



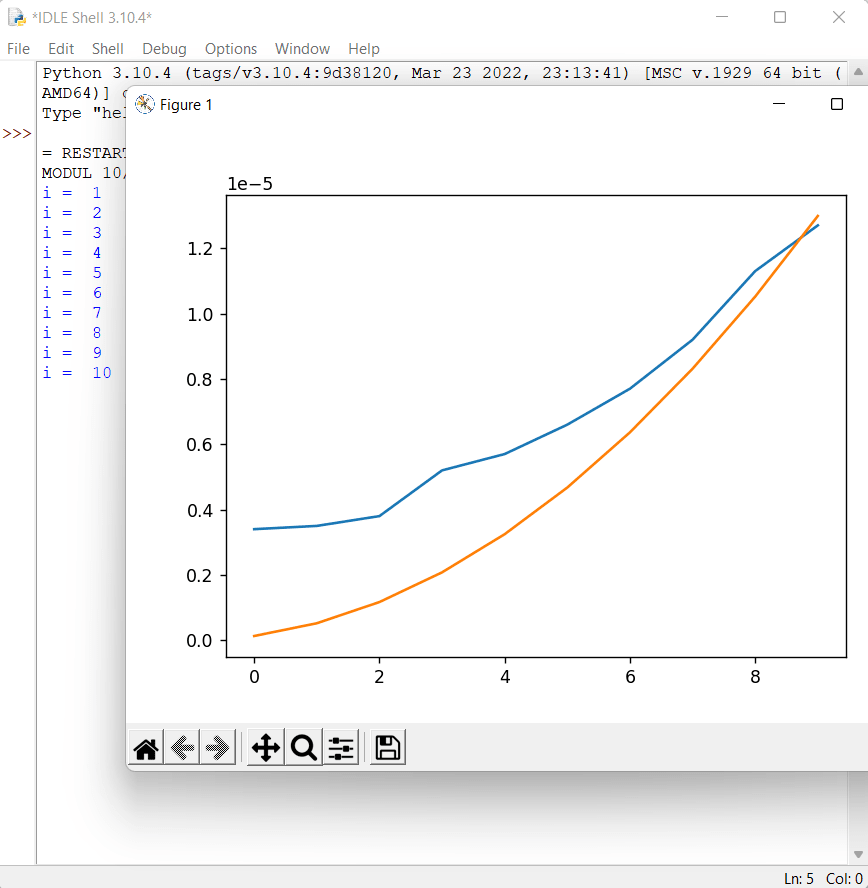
Output



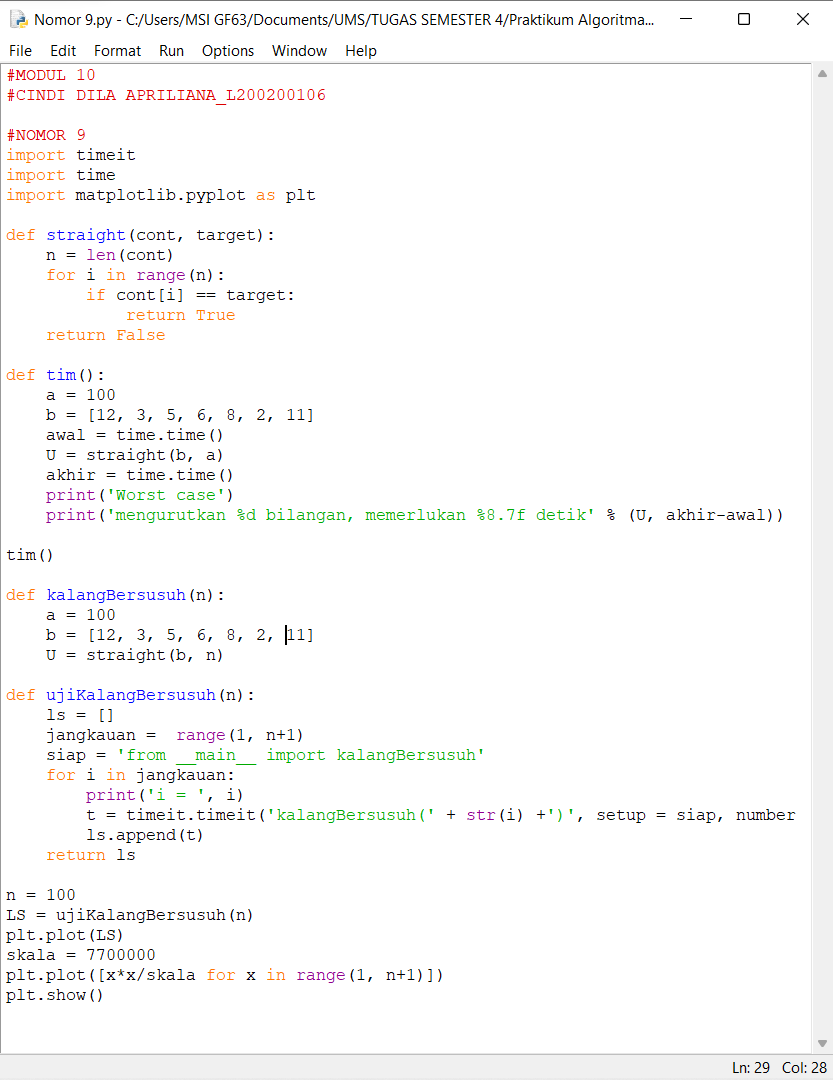
1. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode insert() adalah O(n).



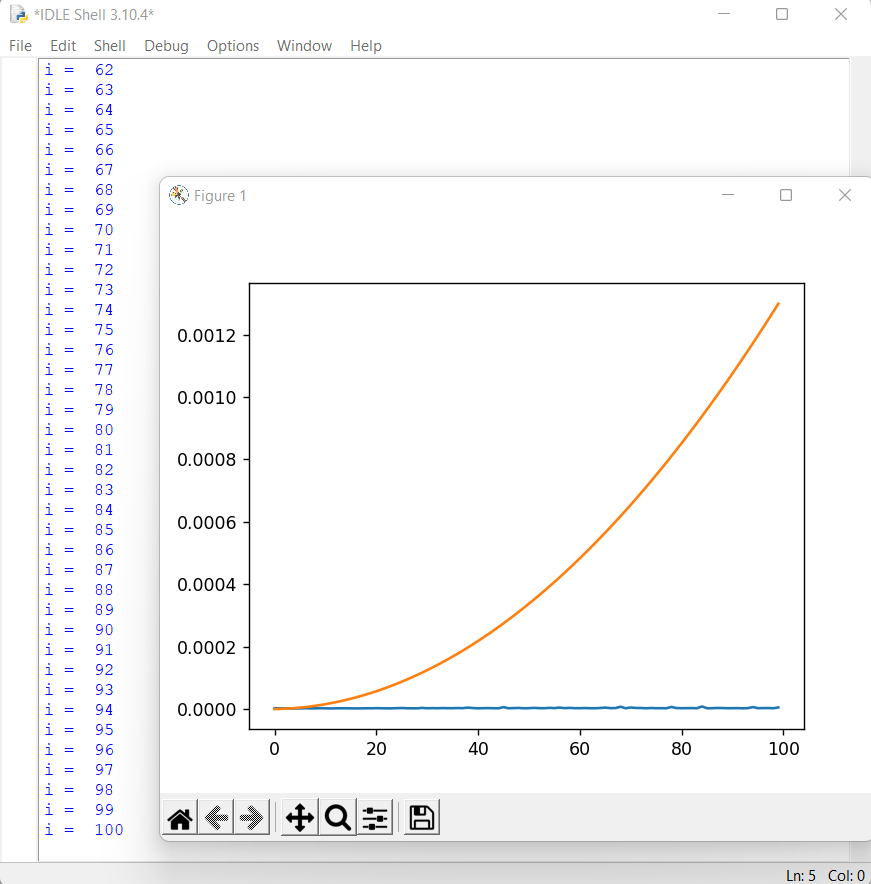
Output



1. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah-suatu-nilai-berada-di-suatu-list mempunyai kompleksitas O(n).



Output



1. Carilah di Internet, kompleksitas metode-metode pada object dict di Python

Waktu Kasus Rata-rata yang terdaftar untuk objek dict mengasumsikan bahwa fungsi hash untuk objek cukup kuat untuk membuat tabrakan menjadi tidak biasa. Kasus Rata-rata mengasumsikan kunci yang digunakan dalam parameter dipilih secara seragam secara acak dari kumpulan semua kunci.

1. Perbedaan notasi big-O , big-Theta dan notasi big-Omega:

* 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.

1. 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 untuk melakukan satu urutan operasi pada struktur data terhadap keseluruhan operasi yang dilakukan