

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

```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py
- jumlahkan_cara_1 -
Jumlah adalah 1, memerlukan 0.00012230 detik
Jumlah adalah 1, memerlukan -0.00129000 detik
Jumlah adalah 1, memerlukan -0.00021140 detik
Jumlah adalah 1, memerlukan -0.00055940 detik
Jumlah adalah 1, memerlukan -0.00668910 detik
>>>
```

```
no_1.py - D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py (3.7.7)
File Edit Format Run Options Window Help
import timeit
import random

print("- jumlahkan_cara_1 -")
def jumlahkan_cara_1(n):
    hasilnya = 0
    for i in range(1, n+1):
        hasilnya = hasilnya + i
    return hasilnya

for i in range(5): # mengulang lima kali
    awal = timeit.timeit() # menandai awal kerja
    h = jumlahkan_cara_1(10000) # menjumlah 1 sampai sepuluh ribu
    akhir = timeit.timeit() # menandai akhir kerja, lalu mencetak
    print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, akhir-awal))
```

b Jumlahkan_cara_2

```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py
- jumlahkan_cara_2 -
Jumlah adalah 50005000, memerlukan -0.00866650 detik
Jumlah adalah 50005000, memerlukan 0.01200800 detik
Jumlah adalah 50005000, memerlukan -0.00208590 detik
Jumlah adalah 50005000, memerlukan -0.00645320 detik
Jumlah adalah 50005000, memerlukan 0.00495030 detik
>>>
```

```
no_1.py - D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py (3.7.7)
File Edit Format Run Options Window Help
import timeit
import random

print("- jumlahkan_cara_1 -")
def jumlahkan_cara_1(n):
    hasilnya = 0
    for i in range(1, n+1):
        hasilnya = hasilnya + i
    return hasilnya

for i in range(5): # mengulang lima kali
    awal = timeit.timeit() # menandai awal kerja
    h = jumlahkan_cara_1(10000) # menjumlah 1 sampai sepuluh ribu
    akhir = timeit.timeit() # menandai akhir kerja, lalu mencetak
    print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, akhir-awal))

print("\n")
print("- jumlahkan_cara_2 -")

def jumlahkan_cara_2(n):
    return (n*(n+1))/2

for i in range(5): # mengulang lima kali
    awal = timeit.timeit() # menandai awal kerja
    h = jumlahkan_cara_2(10000) # menjumlah 1 sampai sepuluh ribu
    akhir = timeit.timeit() # menandai akhir kerja, lalu mencetak
    print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, akhir-awal))
```

c insertionSort

```
Python 3.7.7 Shell
File Edit Shell Debug Options Window Help
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py =====
-----Insertion Sort-----
- average case scenario -
Mengurutkan 3000 bilangan, memerlukan -0.0079073 detik
Mengurutkan 3000 bilangan, memerlukan -0.0088505 detik
Mengurutkan 3000 bilangan, memerlukan -0.0017808 detik
Mengurutkan 3000 bilangan, memerlukan 0.0072728 detik
Mengurutkan 3000 bilangan, memerlukan 0.0034034 detik
=====
- worst case scenario -
Mengurutkan 3000 bilangan, memerlukan 0.0038061 detik
Mengurutkan 3000 bilangan, memerlukan -0.0004260 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001958 detik
Mengurutkan 3000 bilangan, memerlukan -0.0115230 detik
Mengurutkan 3000 bilangan, memerlukan -0.0037518 detik
=====
- best case scenario -
Mengurutkan 3000 bilangan, memerlukan 0.0051893 detik
>>>
```

```
no_1.py - D:/Semester 4/Praktikum ASD/Modul 10/Tugas Modul 10/no_1.py (3.7.7)
File Edit Format Run Options Window Help
.....
nilai = A[i]
pos = i
while pos > 0 and nilai < A[pos-1]:
    A[pos] = A[pos-1]
    pos = pos-1
A[pos] = nilai

print("\n")
print("-----Insertion Sort-----")
print("- average case scenario -")

for i in range(5):
    L = list(range(3000))
    random.shuffle(L) # Mengacak posisi elemen di list
    awal = timeit.timeit()
    U = insertionSort(L)
    akhir = timeit.timeit()
    print("Mengurutkan %d bilangan, memerlukan %9.7f detik" % (len(L),akhir-awal))

print("-----")
print("\n")
print("- worst case scenario -")

for i in range(5):
    L = list(range(3000))
    L = L[::-1] # Membalik urutan elemen di list
    awal = timeit.timeit()
    U = insertionSort(L)
    akhir = timeit.timeit()
    print("Mengurutkan %d bilangan, memerlukan %9.7f detik" % (len(L),akhir-awal))

print("-----")
print("\n")
print("- best case scenario -")

for i in range(5):
    L = list(range(3000))

    awal = timeit.timeit()
    U = insertionSort(L)
    akhir = timeit.timeit()
    print("Mengurutkan %d bilangan, memerlukan %9.7f detik" % (len(L),akhir-awal))
```

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

```

# worst case
def worstcase():
    print("-----Worst Case-----")
    for i in range(1):
        L=list(range(3000))
        L=L[::-1]
        awal=timeit.timeit()
        U=sorted(L)
        akhir=timeit.timeit()
        print("mengurutkan %d bilangan,memerlukan waktu %8.7f detik" %(len(L), akhir-awal))

g = [13, 7, 5, 29, 19]      ## List Urut\
print("g = ", g)
g = sorted(g)
print("-----Data urut")
print("g = ", g)
bestcase()
averagecase()
worstcase()
print("\n")

z = g[::-1]                ## List data inputan terbalik
print("-----Data terbalik")
print("data terbalik = ", z)
bestcase()
averagecase()
worstcase()
print("\n")

random.shuffle(g)          ## List data g acak (random shuffle)
print("-----Data Acak")
print("data acak = ", g)
bestcase()
averagecase()
worstcase()
print("\n")

```

Python 3.7.7 Shell

```

>>>
===== RESTART: D:\Semester 4\Praktikum ASD\Modul 10\Tugas Modul 10\n0_2.py =====
g = [13, 7, 5, 29, 19]
g = [5, 7, 13, 19, 29]
-----Data urut-----
Best Case=====
mengurutkan 3000 bilangan,memerlukan waktu -0.0020653 detik
Average Case=====
mengurutkan 3000 bilangan,memerlukan waktu -0.0026603 detik
Worst Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0005467 detik

-----Data terbalik
data terbalik = [29, 19, 13, 7, 5]
Best Case=====
mengurutkan 3000 bilangan,memerlukan waktu -0.0035716 detik
Average Case=====
mengurutkan 3000 bilangan,memerlukan waktu -0.0013952 detik
Worst Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0022786 detik

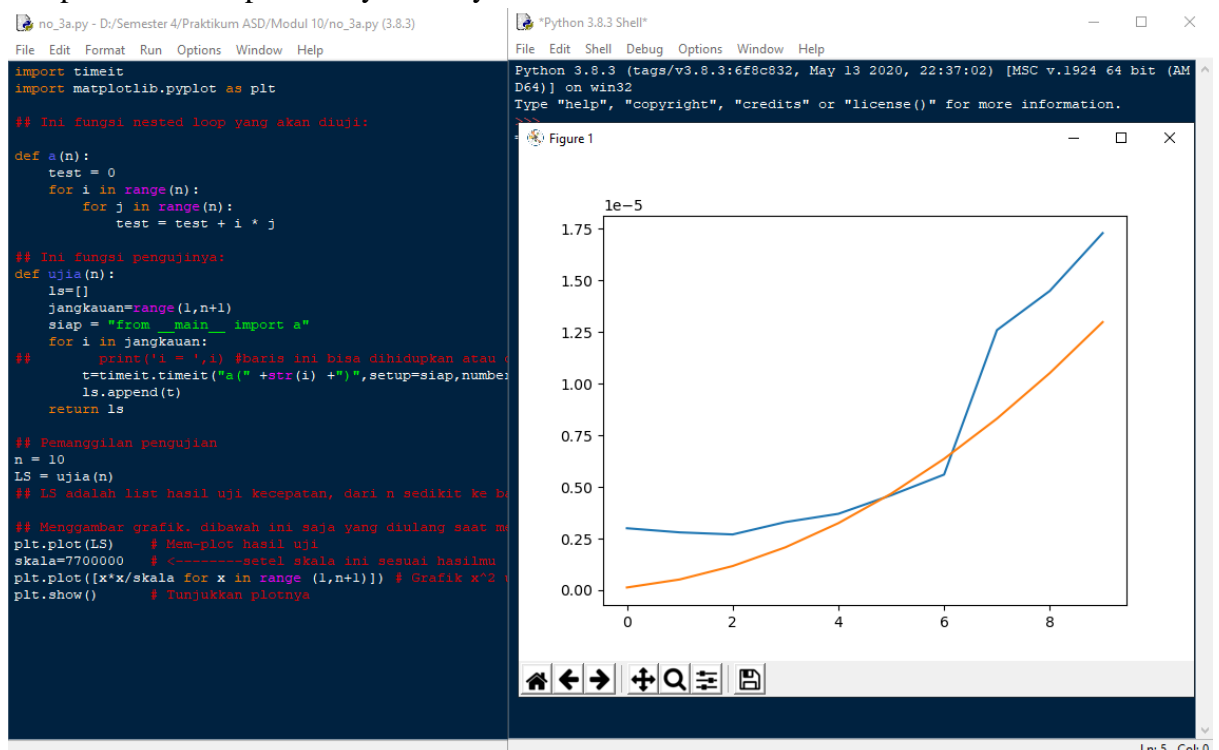
-----Data Acak
data acak = [29, 13, 5, 19, 7]
Best Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0026961 detik
Average Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0003203 detik
Worst Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0053250 detik

>>>
>>>
>>>
>>>
>>>

```

- 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:



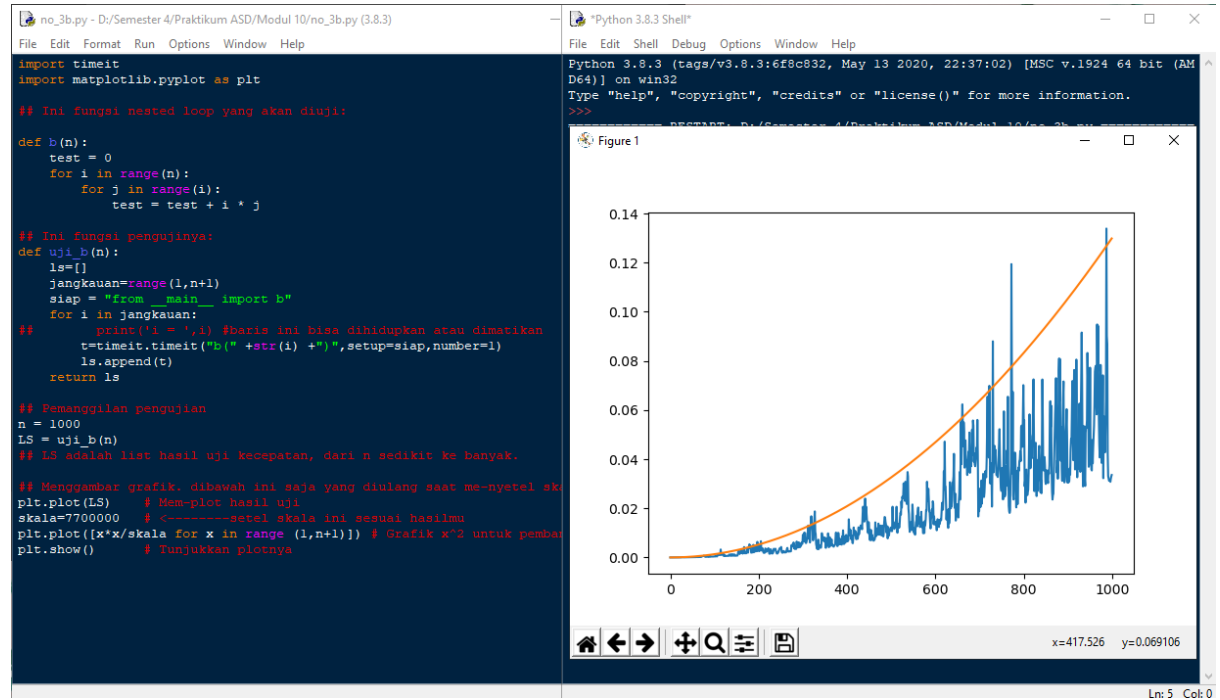
$$T(n) = c1 + n*(n)$$

$$T(n) = c1 + n^2$$

$$O(n) \approx n^2$$

$$O(n^2)$$

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



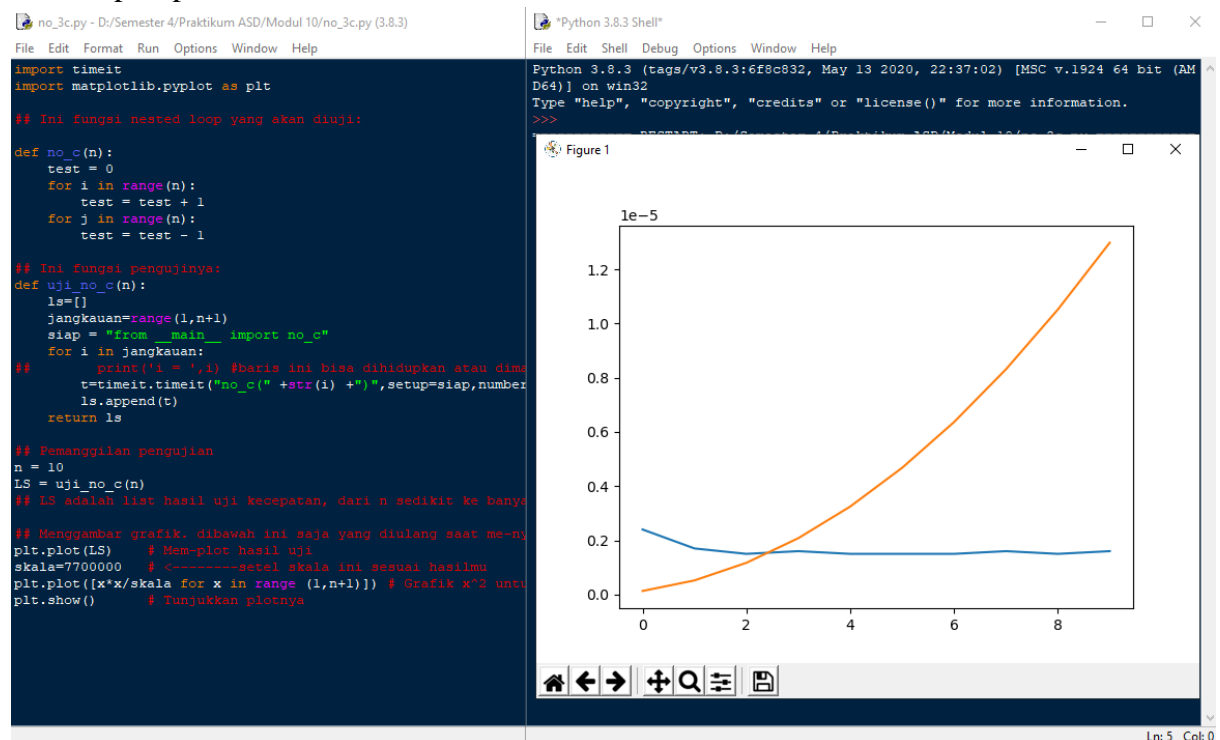
$$T(n) = c1 + \log n$$

$$T(n) = c1 + \log n$$

$$O(n) \approx \log n$$

$$O(\log n)$$

- c Dua loop terpisah



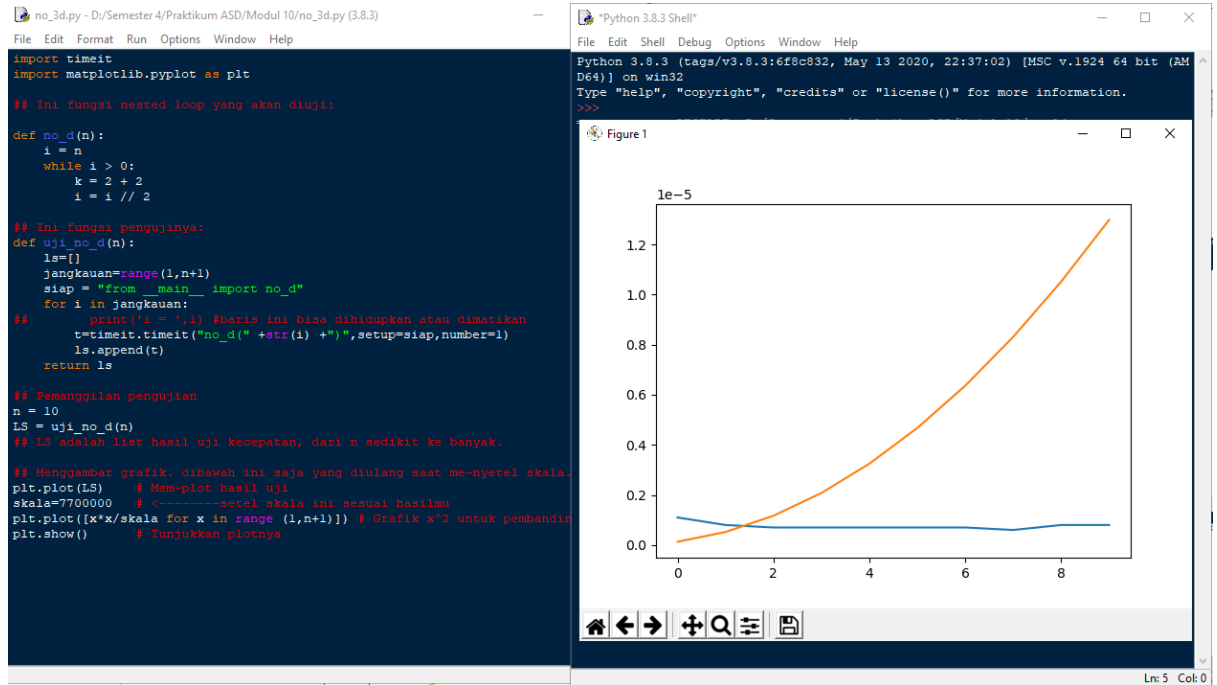
$$T(n) = c1(n) + c2(n)$$

$$T(n) = n + n$$

$$O(n) \approx n$$

$$O(n)$$

d While loop yang dipangkas sepuluh tiap putaran

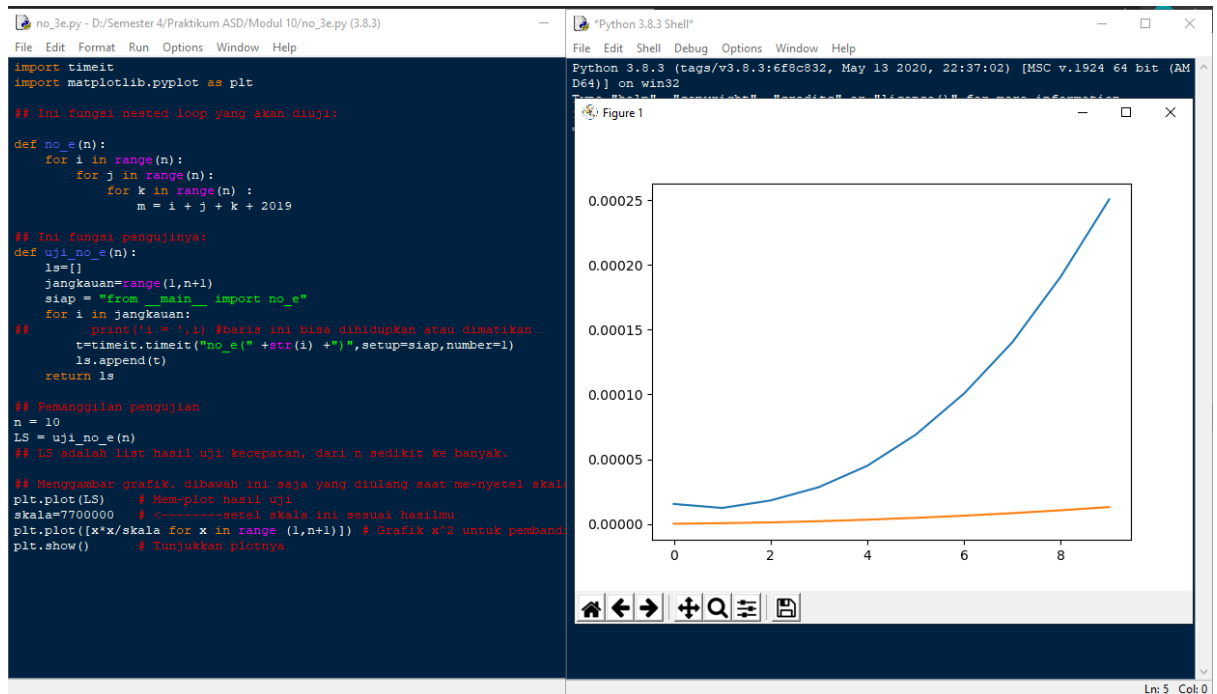


$$T(n) = c_1(1) + c_2(1)$$

$$O(n) \approx 1$$

$$O(1)$$

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



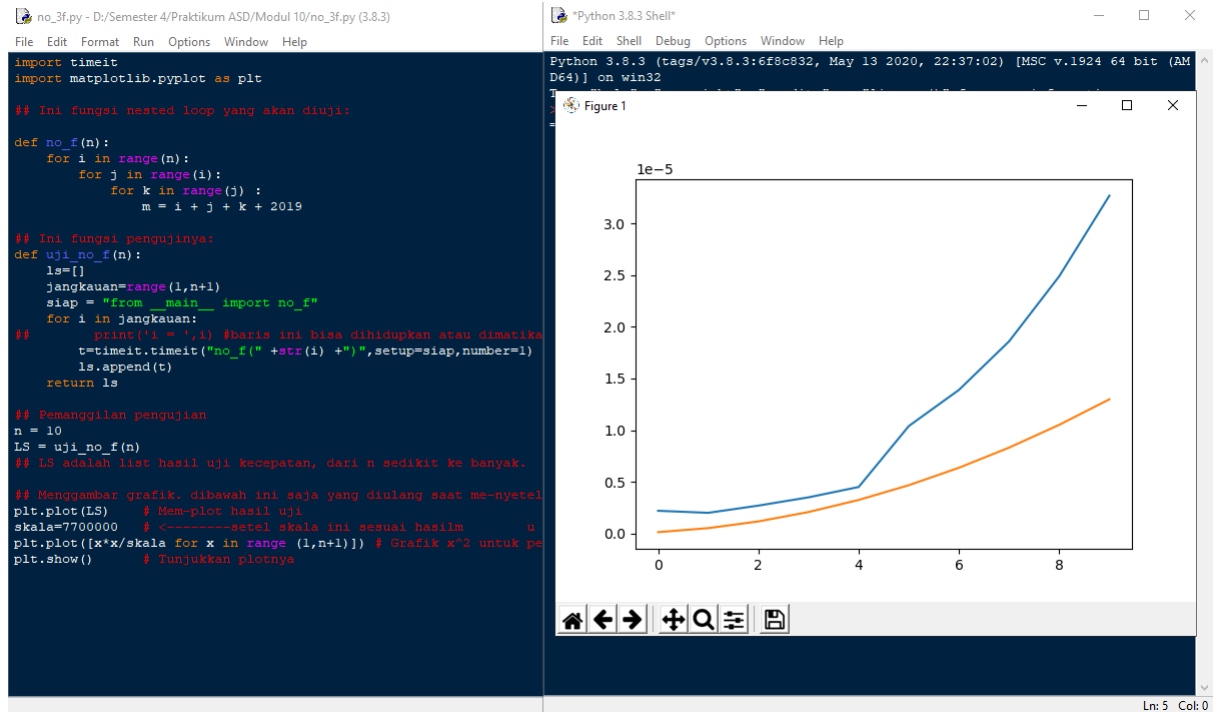
$$T(n) = n \cdot (n \cdot (n))$$

$$T(n) = n^3$$

$$O(n) \approx n^3$$

$$O(n^3)$$

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



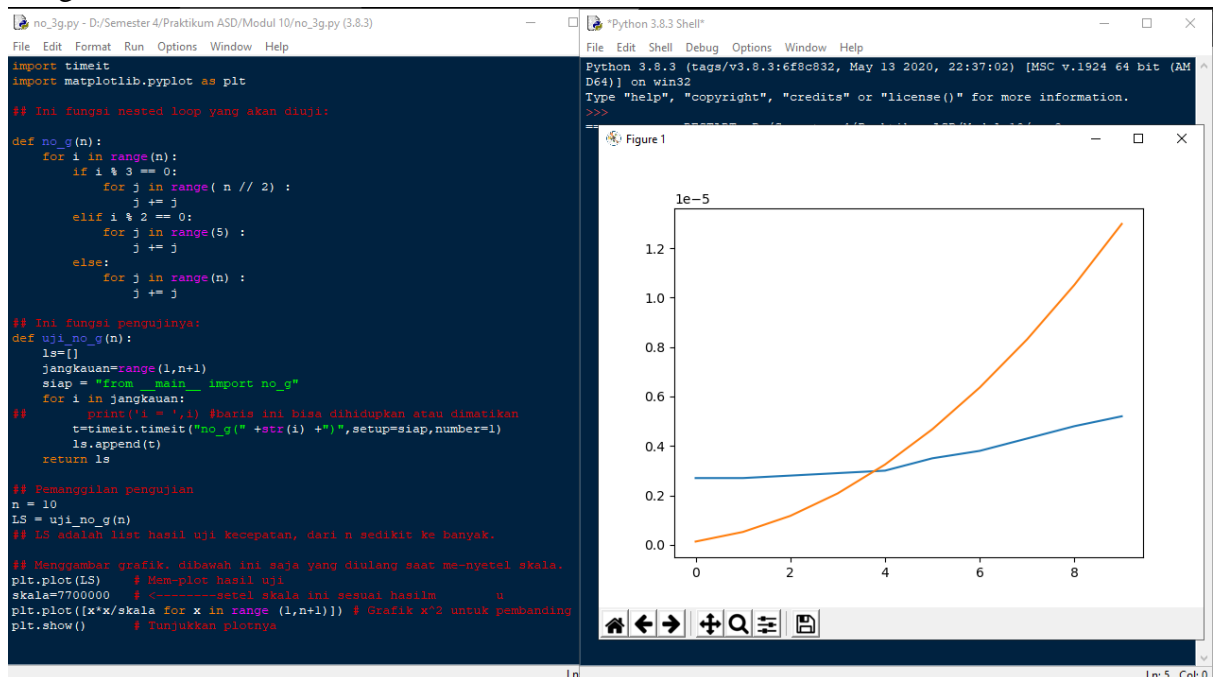
$$T(n) = c_1(n) + c_2(n) + c_3(n)$$

$$T(n) = n + n + n$$

$$O(n) \approx n$$

$$O(n)$$

g Fungsi ini



$$O(n \log n)$$

4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat

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

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

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

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

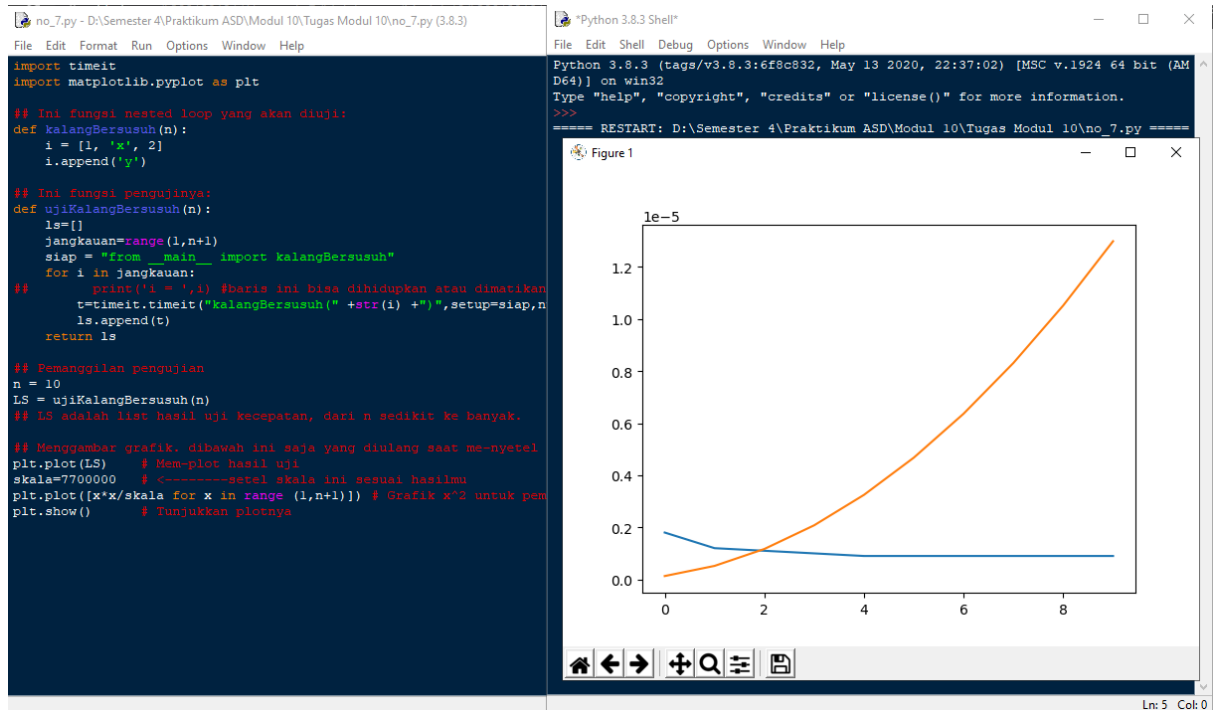
c	$T(n) = 4n + 5n \log n + 102$	$= O(n \log n)$
d	$T(n) = \log n + 3n^2 + 88$	$= O(n^2)$
e	$T(n) = 3(2^n) + n^2 + 647$	$= O(2^n)$
f	$T(n, k) = kn + \log k$	$= O(kn)$
g	$T(n, k) = 8n + k \log n + 800$	$= O(n)$
h	$T(n, k) = 100kn + n$	$= O(kn)$

6. Carilah di internet, kompleksitas metode pada object list di python.

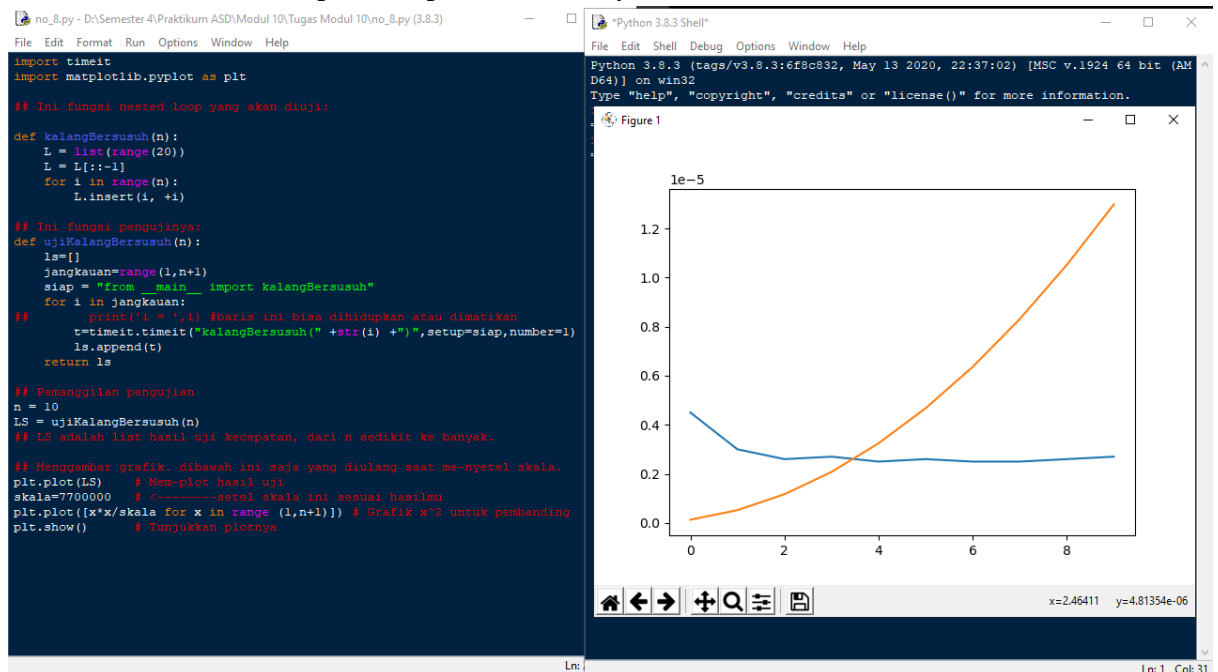
- Google python list method complexity. Lihat juga bagian “Images” –nya
- Kunjungi <https://wiki.python.org/moin/TimeComplexity>

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	$O(1)$	$O(1)$

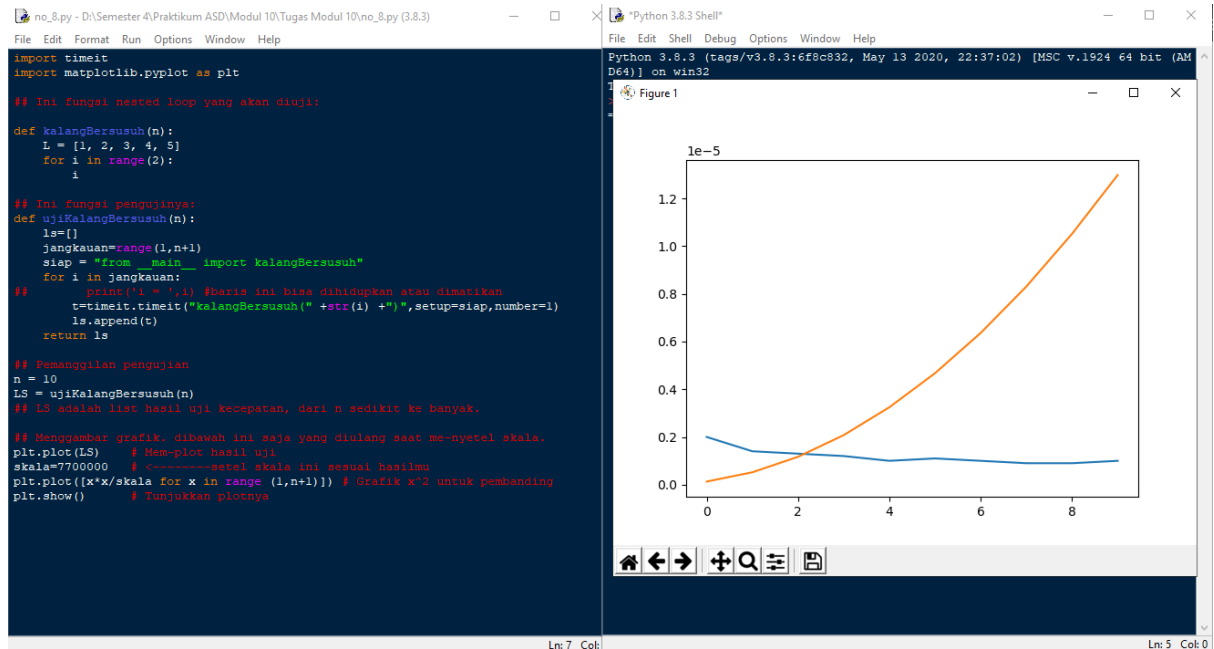
7. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode `append()` adalah $O(1)$.
Gunakan `timeit` dan `matplotlib` seperti sebelumnya.



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



10. Carilah di internet, kompleksitas 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 $\Theta(.)$ dan notasi big-Omega $\Omega(.)$. Apakah beda diantara ketiganya?

- Big O dilambangkan dengan notasi $O(...)$ merupakan keadaan terburuk (worst case). Kinerja sebuah algoritma biasanya diukur menggunakan patokan keadaan Big-O ini. Merupakan notasi asymptotic untuk batas fungsi dari atas dan bawah dengan Berperilaku mirip dengan \leq operator untuk tingkat pertumbuhan.
- Big Theta dilambangkan dengan notasi $\Theta(...)$ 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 $\Omega(...)$ merupakan notasi asymptotic untuk batas bawah dengan keadaan rata-rata (average case) yang berperilaku mirip dengan \geq 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 untuk melakukan satu urutan operasi pada struktur data terhadap keseluruhan operasi yang dilakukan.