

**PRAKTIKUM ALGORITMA DAN
STRUKTUR DATA**

Modul 10

Analisi Algoritma



Disusun oleh:

DONI WAHYU SAPUTRO

L200200169

G

PROGRAM STUDI TEKNIK INFORMATIKA

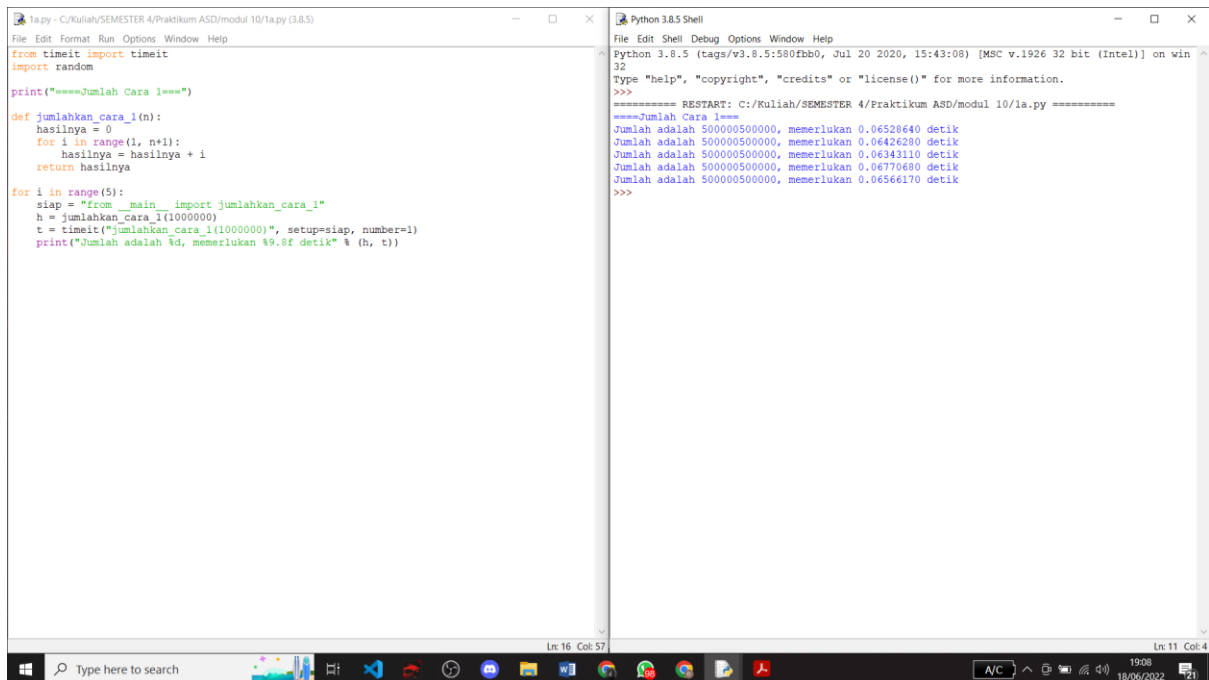
FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

Latihan

1.

a



```
1a.py - C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1a.py (3.8.5)
File Edit Format Run Options Window Help
from timeit import timeit
import random

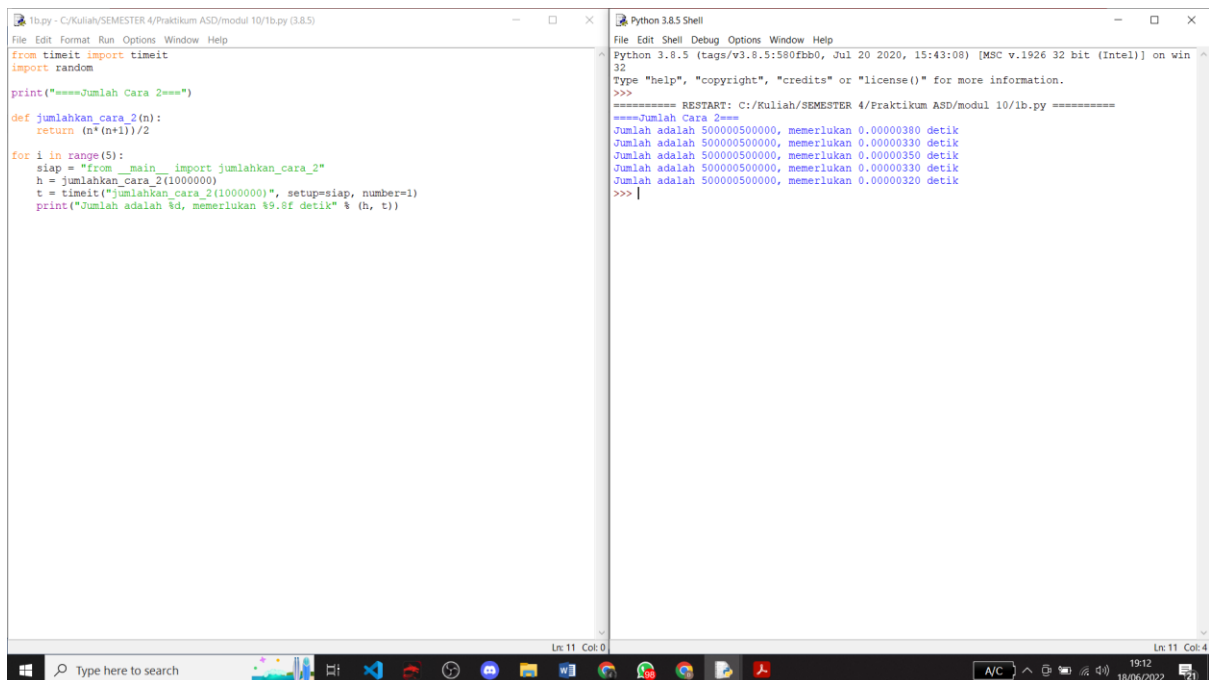
print("====Jumlah Cara 1====")

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

for i in range(5):
    siap = "from __main__ import jumlahkan_cara_1"
    h = jumlahkan_cara_1(1000000)
    t = timeit("jumlahkan_cara_1(1000000)", setup=siap, number=1)
    print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, t))

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1a.py =====
====Jumlah Cara 1====
Jumlah adalah 500000500000, memerlukan 0.06528640 detik
Jumlah adalah 500000500000, memerlukan 0.06426280 detik
Jumlah adalah 500000500000, memerlukan 0.06343110 detik
Jumlah adalah 500000500000, memerlukan 0.06770680 detik
Jumlah adalah 500000500000, memerlukan 0.06566170 detik
>>>
```

b



```
1b.py - C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1b.py (3.8.5)
File Edit Format Run Options Window Help
from timeit import timeit
import random

print("====Jumlah Cara 2====")

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

for i in range(5):
    siap = "from __main__ import jumlahkan_cara_2"
    h = jumlahkan_cara_2(1000000)
    t = timeit("jumlahkan_cara_2(1000000)", setup=siap, number=1)
    print("Jumlah adalah %d, memerlukan %9.8f detik" % (h, t))

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1b.py =====
====Jumlah Cara 2====
Jumlah adalah 500000500000, memerlukan 0.00000380 detik
Jumlah adalah 500000500000, memerlukan 0.00000330 detik
Jumlah adalah 500000500000, memerlukan 0.00000350 detik
Jumlah adalah 500000500000, memerlukan 0.00000330 detik
Jumlah adalah 500000500000, memerlukan 0.00000320 detik
>>>
```

C

```

1c.py - C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1c.py (3.8.5)
File Edit Format Run Options Window Help
from timeit import timeit
import random

print("====Insertion Sort====")

def insertionSort(A):
    n = len(A)
    for i in range(1, n):
        nilai = A[i]
        pos = i
        while pos > 0 and nilai < A[pos - 1]:
            A[pos] = A[pos - 1]
            pos = pos - 1
        A[pos] = nilai

for i in range(5):
    siap = "from __main__ import insertionSort,L"
    L = list(range(3000))
    t = timeit("insertionSort(L)", setup=siap, number=1)
    print("Jumlah adalah %d bilangan, memerlukan %9.7f detik" % (len(L), t))
  
```

```

Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/1c.py =====
====Insertion Sort====
Jumlah adalah 3000 bilangan, memerlukan 0.0021099 detik
Jumlah adalah 3000 bilangan, memerlukan 0.0021175 detik
Jumlah adalah 3000 bilangan, memerlukan 0.0013292 detik
Jumlah adalah 3000 bilangan, memerlukan 0.0013251 detik
Jumlah adalah 3000 bilangan, memerlukan 0.0013169 detik
>>>
  
```

2.

```

2.py - C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/2.py (3.8.5)
File Edit Format Run Options Window Help
from timeit import timeit
import random

print("====Sorted avg case====")

for i in range(5):
    g = list(range(3000))
    random.shuffle(g)
    t = timeit("sorted(g)", setup="from __main__ import g,number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(g), t))

print("====Sorted best case====")

for i in range(5):
    g = list(range(3000))
    t = timeit("sorted(g)", setup="from __main__ import g,number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(g), t))

print("====Sorted worst case====")

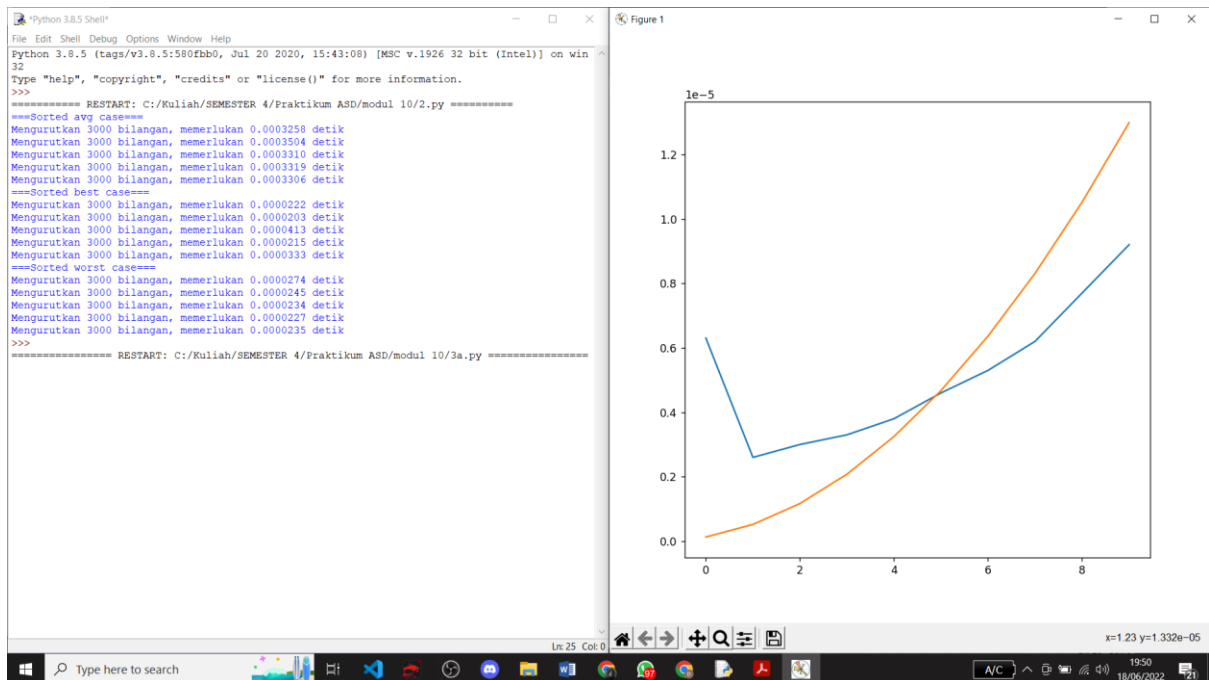
for i in range(5):
    g = list(range(3000))
    g = g[::-1]
    t = timeit("sorted(g)", setup="from __main__ import g,number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(g), t))
  
```

```

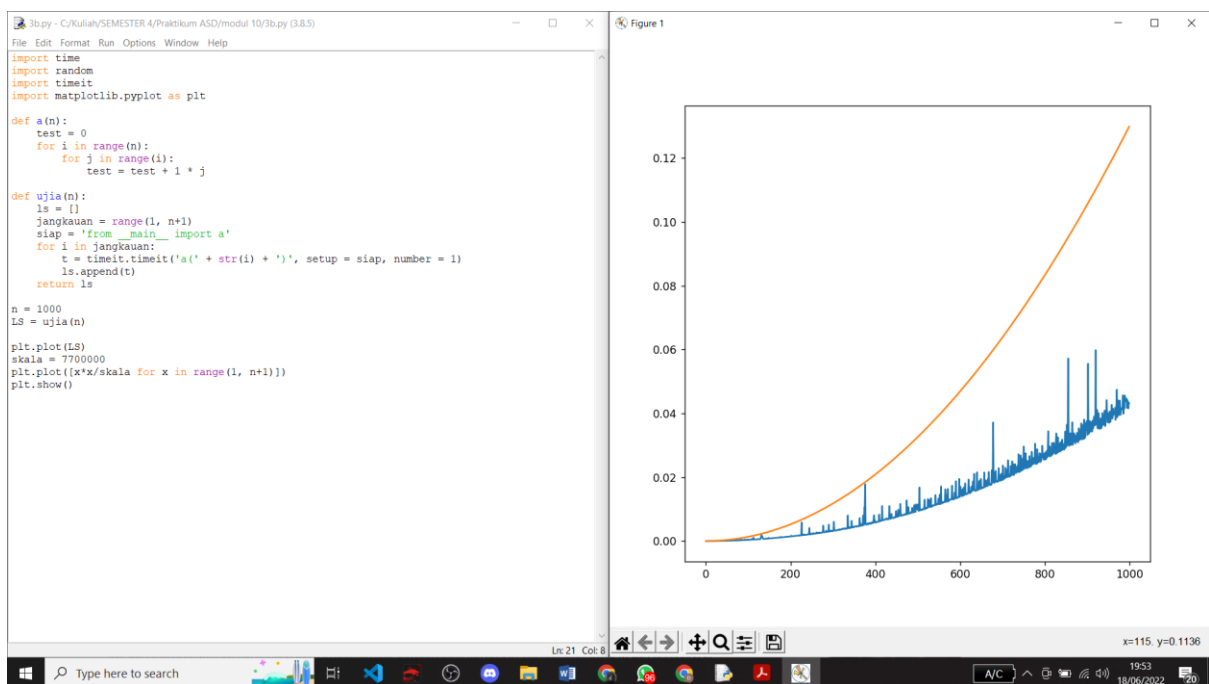
Python 3.8.5 Shell
File Edit Shell Debug Options Window Help
Python 3.8.5 (tags/v3.8.5:580fbb0, Jul 20 2020, 15:43:08) [MSC v.1926 32 bit (Intel)] on win
32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Kuliah/SEMESTER 4/Praktikum ASD/modul 10/2.py =====
====Sorted avg case====
Mengurutkan 3000 bilangan, memerlukan 0.0003258 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003504 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003310 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003319 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003306 detik
====Sorted best case====
Mengurutkan 3000 bilangan, memerlukan 0.0000222 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000203 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000413 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000215 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000333 detik
====Sorted worst case====
Mengurutkan 3000 bilangan, memerlukan 0.0000274 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000245 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000234 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000227 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000235 detik
>>>
  
```

3.

a



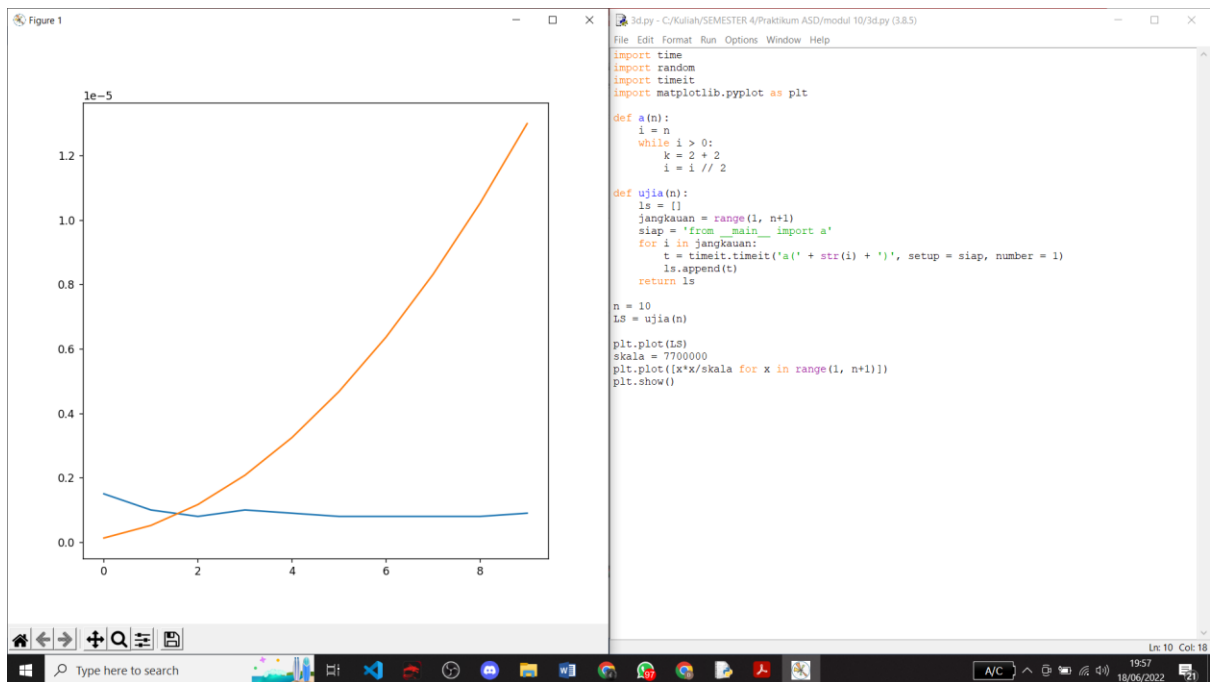
b



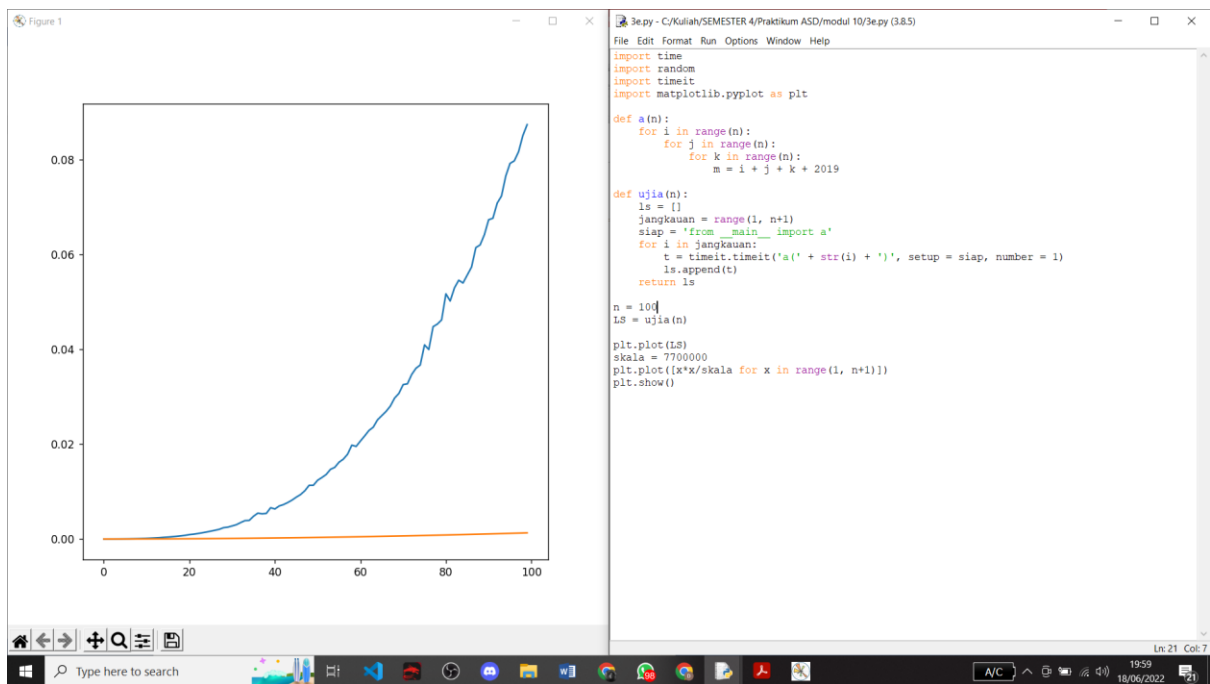
c



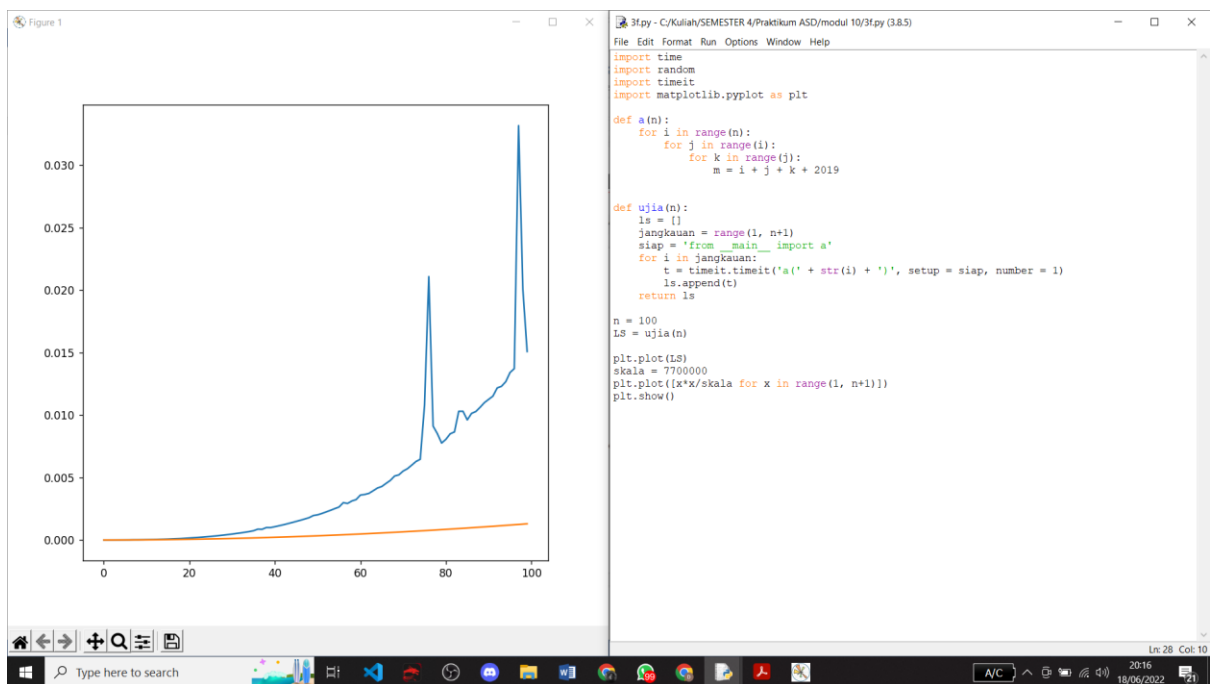
d



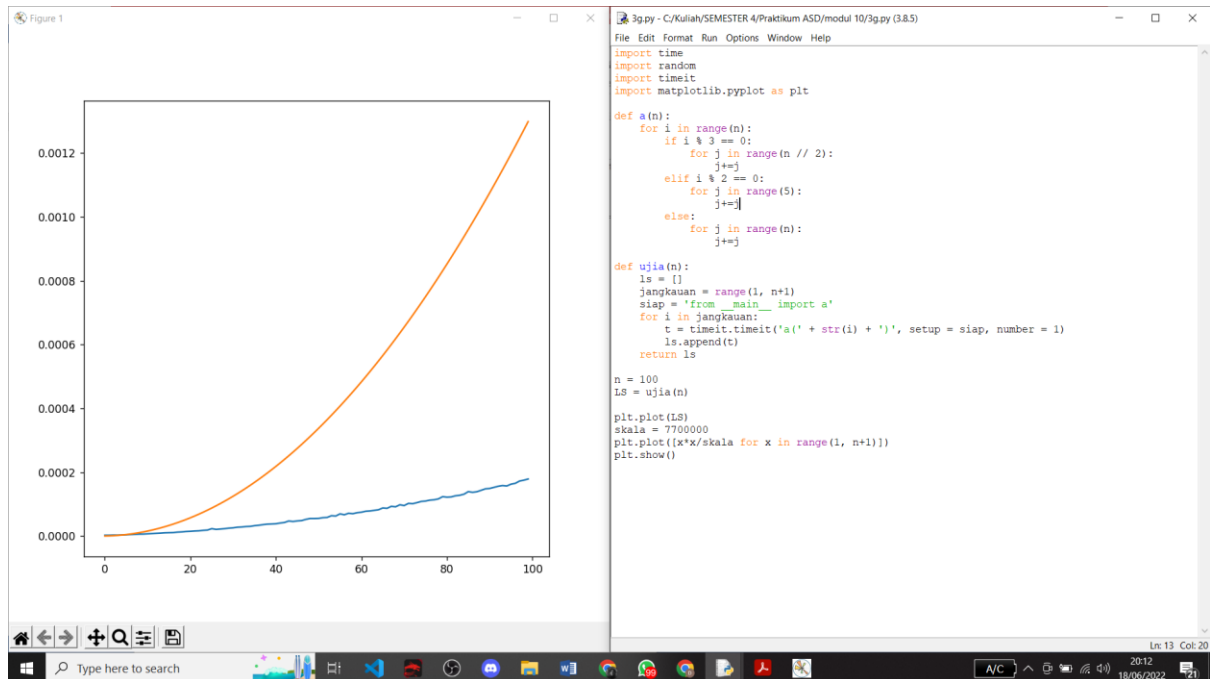
e



f



g



4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat $\log 4n < 10\log 2n < n$
 $\log 2n < 2 \log 2n < 5n^2 < n^3 < 12n^6 < 4n$

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

- $T(n) = n^2 + 32n + 8 = O(n^2)$
- $T(n) = 87n + 8n = O(n)$
- $T(n) = 4n + 5n \log n + 102 = O(n \log n)$
- $T(n) = \log n + 3n^2 + 88 = O(n^2)$
- $T(n) = 3(2^n) + n^2 + 647 = O(2^n)$
- $T(n, k) = kn + \log k = O(kn)$
- $T(n, k) = 8n + k \log n + 800 = O(n)$
- $T(n, k) = 100kn + n = O(kn)$

6. (Literatur 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>

wiki.python.org/moin/TimeComplexity

list

The Average Case assumes parameters generated uniformly at random.

Internally, a list is represented as an array; the largest costs come from growing beyond the current allocation size (because everything must move), or from inserting or deleting somewhere near the beginning (because everything after that must move). If you need to add/remove at both ends, consider using a collections.deque instead.

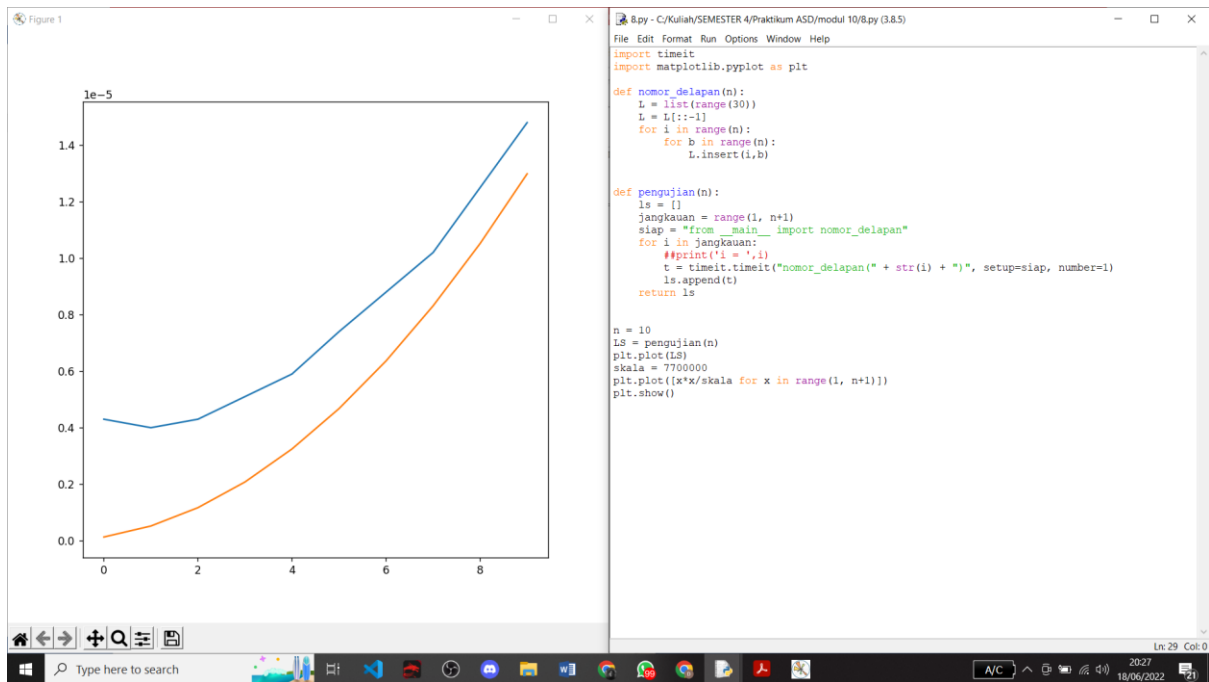
Operation	Average Case	Amortized Worst Case
Copy	$O(n)$	$O(n)$
Append[1]	$O(1)$	$O(1)$
Pop last	$O(1)$	$O(1)$
Pop intermediate[2]	$O(n)$	$O(n)$
Insert	$O(n)$	$O(n)$
Get Item	$O(1)$	$O(1)$
Set Item	$O(1)$	$O(1)$
Delete Item	$O(n)$	$O(n)$
Iteration	$O(n)$	$O(n)$
Get Slice	$O(k)$	$O(k)$
Del Slice	$O(n)$	$O(n)$
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	$O(1)$	$O(1)$

collections.deque

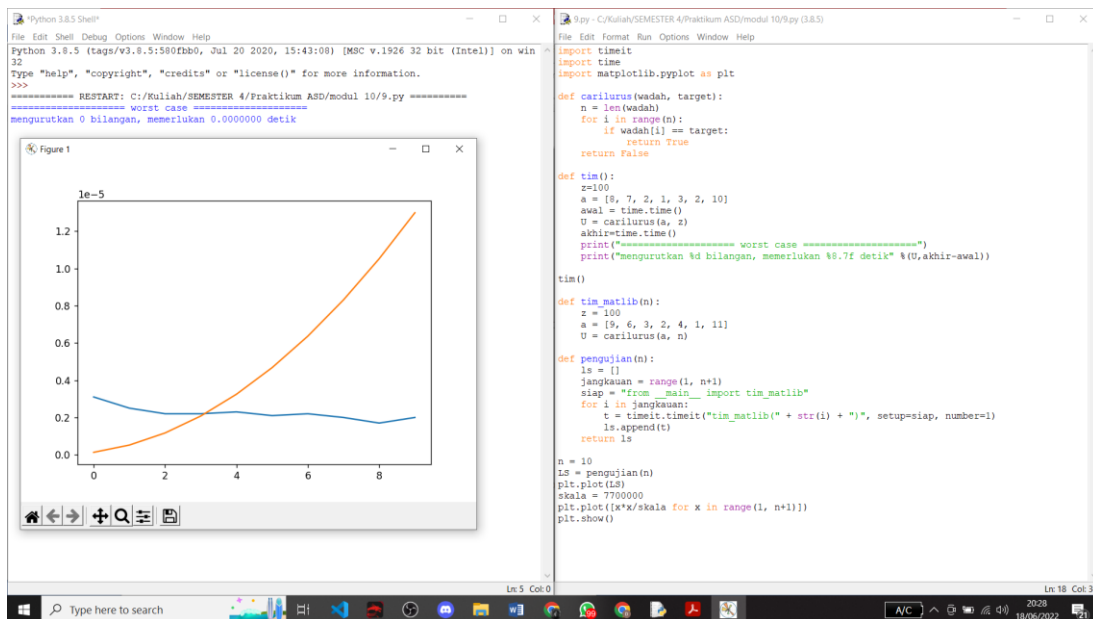
7.



8.



9.



10.

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.

- Big O dilambangkan dengan notasi $O(\dots)$ 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 $\Theta(\dots)$ 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 $\Omega(\dots)$ merupakan notasi asymptotic untuk batas bawah dengan keadaan rata-rata(average case) yang berperilaku mirip dengan \geq operator untuk tingkat pertumbuhan.

12.

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