

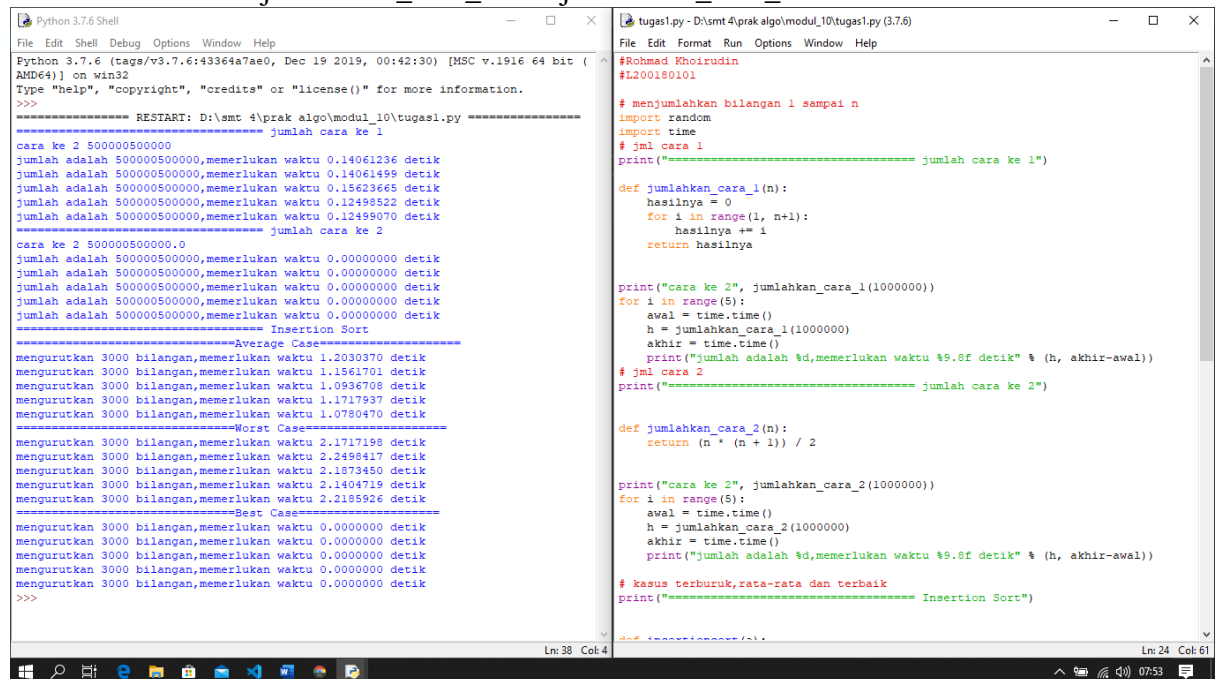
Nama : Rohmad Khoirudin
NIM : L200180101
Kelas : D

Prak-ASD

Modul 10

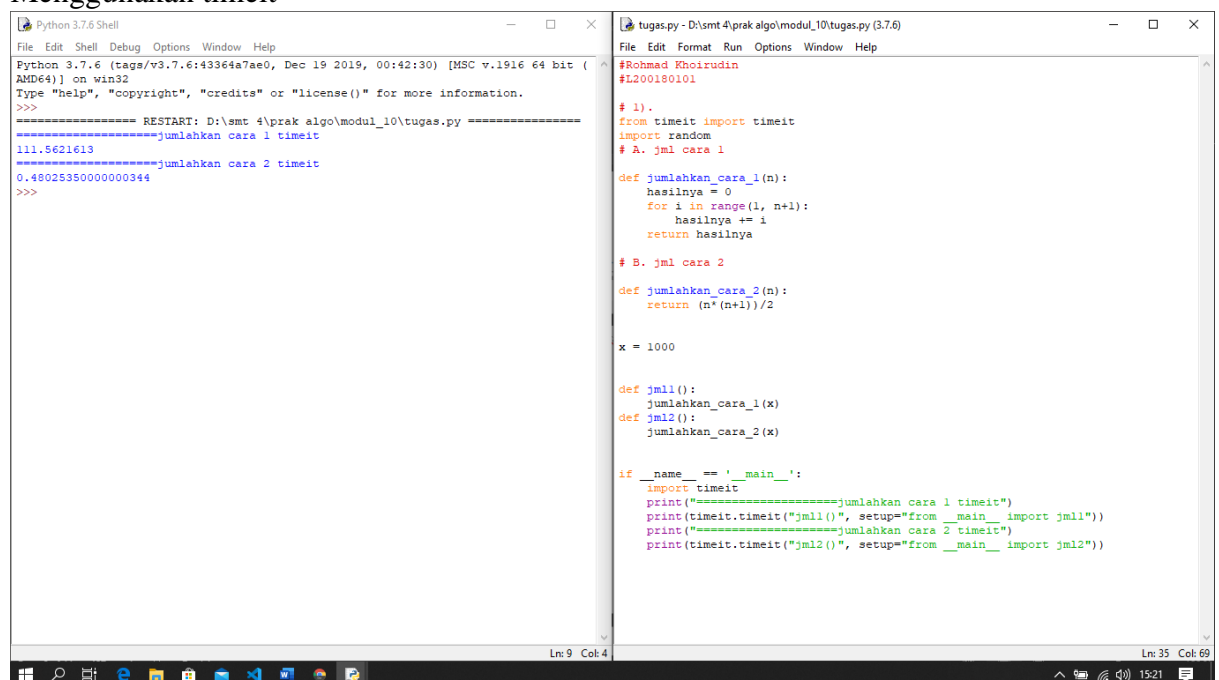
Tugas

- 1) Kerjakan ulang contoh dan Latihan di modul timeit, yakni jumlahkan_cara_1 , jumlahkan_cara_2 dan insertionSort.
- Bestcase scenarion jumlahkan_cara_1 dan jumlahkan_cara_2



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\smt 4\prak algo\modul_10\tugas1.py =====
===== jumlah cara ke 1
cara ke 2 500000500000
jumlah adalah 500000500000,memerlukan waktu 0.14061236 detik
jumlah adalah 500000500000,memerlukan waktu 0.14061499 detik
jumlah adalah 500000500000,memerlukan waktu 0.15623665 detik
jumlah adalah 500000500000,memerlukan waktu 0.12498522 detik
jumlah adalah 500000500000,memerlukan waktu 0.12499070 detik
===== jumlah cara ke 2
cara ke 2 500000500000.0
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
===== Insertion Sort
=====Average Case=====
mengurutkan 3000 bilangan,memerlukan waktu 1.2030370 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.1561701 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0936708 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.1717937 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0790470 detik
=====Worst Case=====
mengurutkan 3000 bilangan,memerlukan waktu 2.1717198 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.2498417 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.1873450 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.1404719 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.2185926 detik
=====Best Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.00000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.00000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.00000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.00000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.00000000 detik
>>>
```

Menggunakan timeit



```
Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\smt 4\prak algo\modul_10\tugas.py =====
=====jumlahkan cara 1 timeit
111.5621613
=====jumlahkan cara 2 timeit
0.48025350000000344
>>>
```

- Untuk insertion Sort kerjakan untuk ketiga kasusnya

The image shows two side-by-side Python IDE windows. The left window, titled 'Python 3.7.6 Shell', displays the execution output of a script. It shows the execution of 'cara ke 1' and 'cara ke 2' for a list of 500,000 elements, followed by a detailed performance analysis of Insertion Sort for 3,000 elements. The analysis includes 'Average Case', 'Worst Case', and 'Best Case' scenarios, each showing the time taken for 3,000 iterations. The right window, titled 'tugas1.py - D:\smt 4\prak algo\modul_10\tugas1.py (3.7.6)', shows the source code of the insertion sort algorithm. The code defines an 'insertionsort(a)' function and includes timing logic to measure the execution time for different cases.

```

Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\smt 4\prak algo\modul_10\tugas1.py =====
===== jumlah cara ke 1
cara ke 2 500000500000
jumlah adalah 500000500000,memerlukan waktu 0.14061236 detik
jumlah adalah 500000500000,memerlukan waktu 0.14061499 detik
jumlah adalah 500000500000,memerlukan waktu 0.15623665 detik
jumlah adalah 500000500000,memerlukan waktu 0.12498522 detik
jumlah adalah 500000500000,memerlukan waktu 0.12499070 detik
===== jumlah cara ke 2
cara ke 2 500000500000.0
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
jumlah adalah 500000500000,memerlukan waktu 0.00000000 detik
===== Insertion Sort
=====Average Case=====
mengurutkan 3000 bilangan,memerlukan waktu 1.2030370 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.1561701 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0936709 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.1717937 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0780470 detik
=====Worst Case=====
mengurutkan 3000 bilangan,memerlukan waktu 2.1717199 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.2498417 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.1873450 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.1404719 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.2185526 detik
=====Best Case=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
>>>

tugas1.py - D:\smt 4\prak algo\modul_10\tugas1.py (3.7.6)
File Edit Format Run Options Window Help
def insertionsort(a):
    for i in range(1, len(a)):
        nilai = a[i]
        b = i
        while b > 0 and nilai < a[b - 1]:
            a[b] = a[b-1]
            b -= 1
        a[b] = nilai

print("=====Average Case=====")
# average case
for i in range(5):
    L = list(range(3000))
    random.shuffle(L)
    awal = time.time()
    U = insertionsort(L)
    akhir = time.time()
    print("mengurutkan %d bilangan,memerlukan waktu %8.7f detik" %
          (len(L), akhir-awal))

# worst case
print("=====Worst Case=====")
for i in range(5):
    L = list(range(3000))
    L = L[::-1]
    awal = time.time()
    U = insertionsort(L)
    akhir = time.time()
    print("mengurutkan %d bilangan,memerlukan waktu %8.7f detik" %
          (len(L), akhir-awal))

# best case
print("=====Best Case=====")
for i in range(5):
    L = list(range(3000))
    awal = time.time()
    U = insertionsort(L)
    akhir = time.time()
    print("mengurutkan %d bilangan,memerlukan waktu %8.7f detik" %
          (len(L), akhir-awal))
  
```

Menggunakan timeit

The image shows two side-by-side Python IDE windows. The left window, titled 'Python 3.7.6 Shell', shows the execution of a script that uses the 'timeit' module to measure the execution time of the 'insertionsort' function. The output shows a single execution time of 3.1880563 seconds. The right window, titled 'tugas.py - D:\smt 4\prak algo\modul_10\tugas.py (3.7.6)', shows the source code of the script. It includes a header with the author's name and ID, a list of test cases, and a function 'inst()' that calls 'insertionsort' and measures its execution time using 'timeit.timeit()'. The script also includes a main block that imports 'timeit' and 'inst()' and prints the execution time.

```

Python 3.7.6 Shell
File Edit Shell Debug Options Window Help
Python 3.7.6 (tags/v3.7.6:43364a7ae0, Dec 19 2019, 00:42:30) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: D:\smt 4\prak algo\modul_10\tugas.py =====
=====insertionSort timeit
3.1880563
>>>

tugas.py - D:\smt 4\prak algo\modul_10\tugas.py (3.7.6)
File Edit Format Run Options Window Help
#Rohmad Khoirudin
#L200180101

# 1).
from timeit import timeit
import random

# C. Insertion Sort

def insertionsort(a):
    for i in range(1, len(a)):
        nilai = a[i]
        b = i
        while b > 0 and nilai < a[b - 1]:
            a[b] = a[b-1]
            b -= 1
        a[b] = nilai

y = [3, 4, 2, 1, 2, 5, 9, 3]

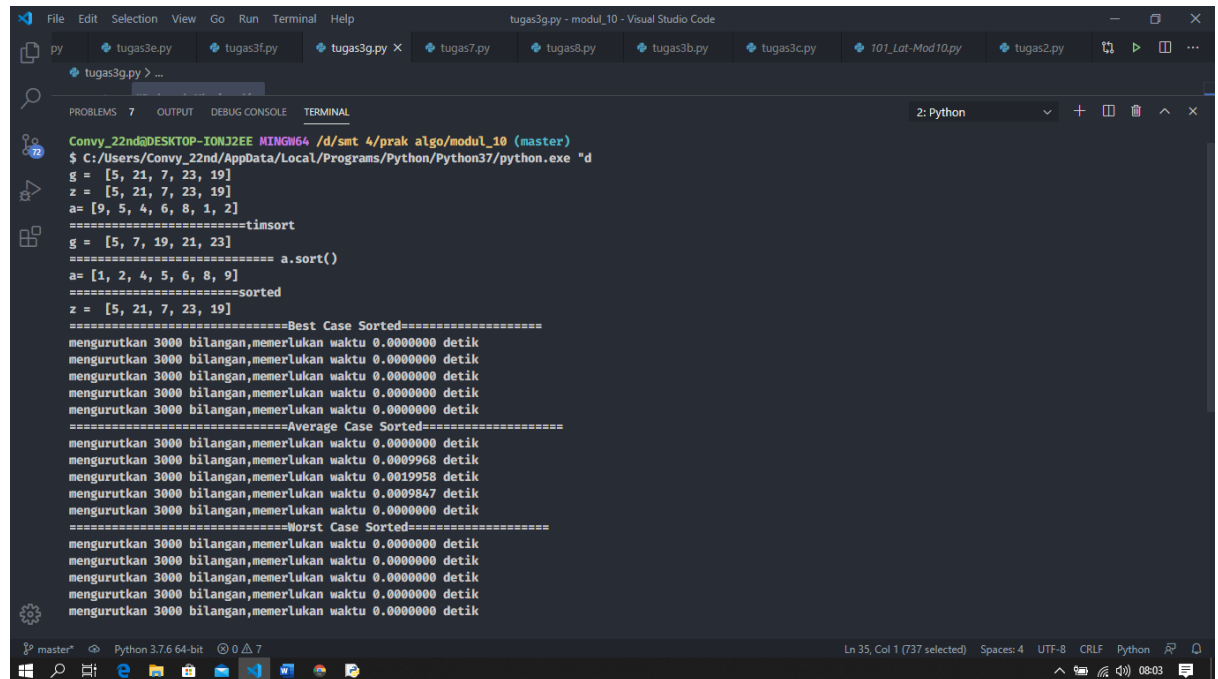
def inst():
    insertionsort(y)

if __name__ == '__main__':
    import timeit
    print("=====insertionSort timeit")
    print(timeit.timeit("inst()", setup="from __main__ import inst"))
  
```

- 2) Tunjukkan program timsort, lalu isi variabel g dengan list angka, urutkan menggunakan g.sort(). Kemudian selidikalah fungsi sorted() menggunakan timeit.

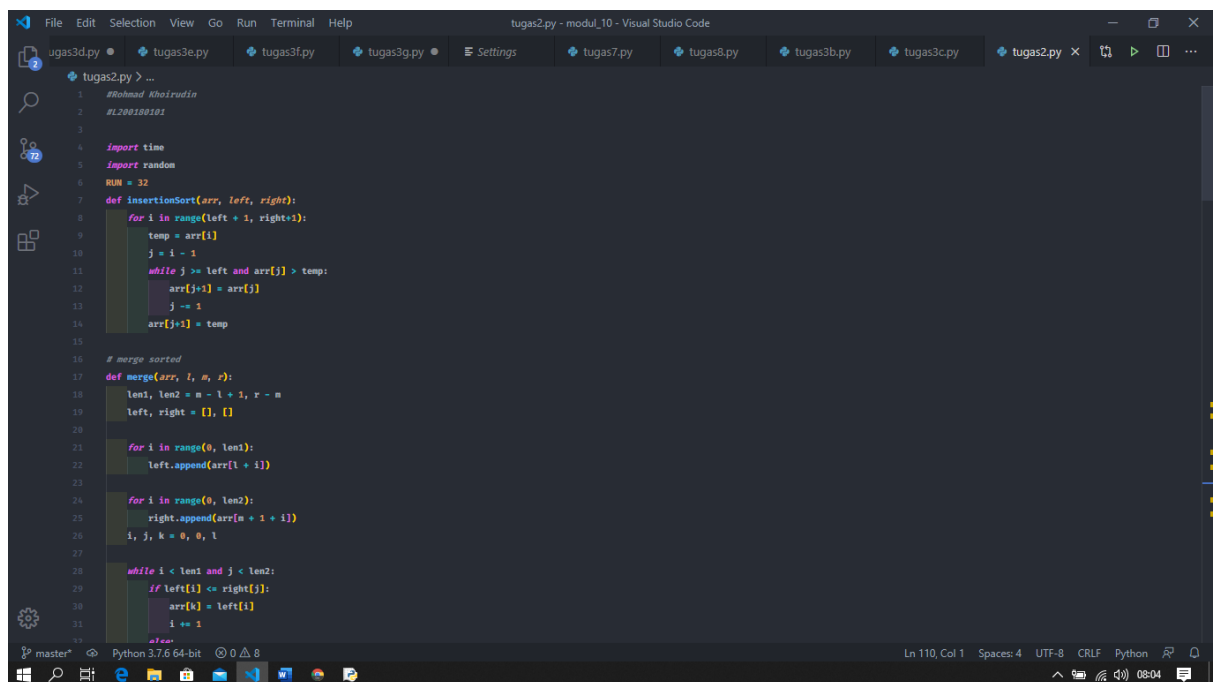
- Bestcase ,averagecase dan Wortcase ke fungsi sorted()

Hasil Run



```
Convy_22nd@DESKTOP-IONJ2EE MINGW64 /d/smt 4/prak algo/modul_10 (master)
$ C:/Users/Convy_22nd/AppData/Local/Programs/Python/Python37/python.exe "d
g = [5, 21, 7, 23, 19]
z = [5, 21, 7, 23, 19]
a = [9, 5, 4, 6, 8, 1, 2]
=====timsort
g = [5, 7, 19, 21, 23]
===== a.sort()
a = [1, 2, 4, 5, 6, 8, 9]
=====sorted
z = [5, 21, 7, 23, 19]
=====Best Case Sorted=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
=====Average Case Sorted=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0009968 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0019958 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0009847 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
=====Worst Case Sorted=====
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0000000 detik
```

Source Code:



```
1 #Rohmad Khoirudin
2 #1200100101
3
4 import time
5 import random
6 RUN = 32
7 def insertionSort(arr, left, right):
8     for i in range(left + 1, right + 1):
9         temp = arr[i]
10        j = i - 1
11        while j >= left and arr[j] > temp:
12            arr[j + 1] = arr[j]
13            j -= 1
14        arr[j + 1] = temp
15
16 # merge sorted
17 def merge(arr, l, m, r):
18     len1, len2 = m - l + 1, r - m
19     left, right = [], []
20
21     for i in range(0, len1):
22         left.append(arr[l + i])
23
24     for i in range(0, len2):
25         right.append(arr[m + 1 + i])
26     i, j, k = 0, 0, l
27
28     while i < len1 and j < len2:
29         if left[i] <= right[j]:
30             arr[k] = left[i]
31             i += 1
32         else:
33             arr[k] = right[j]
34             j += 1
35         k += 1
36
37     while i < len1:
38         arr[k] = left[i]
39         i += 1
40         k += 1
41
42     while j < len2:
43         arr[k] = right[j]
44         j += 1
45         k += 1
```

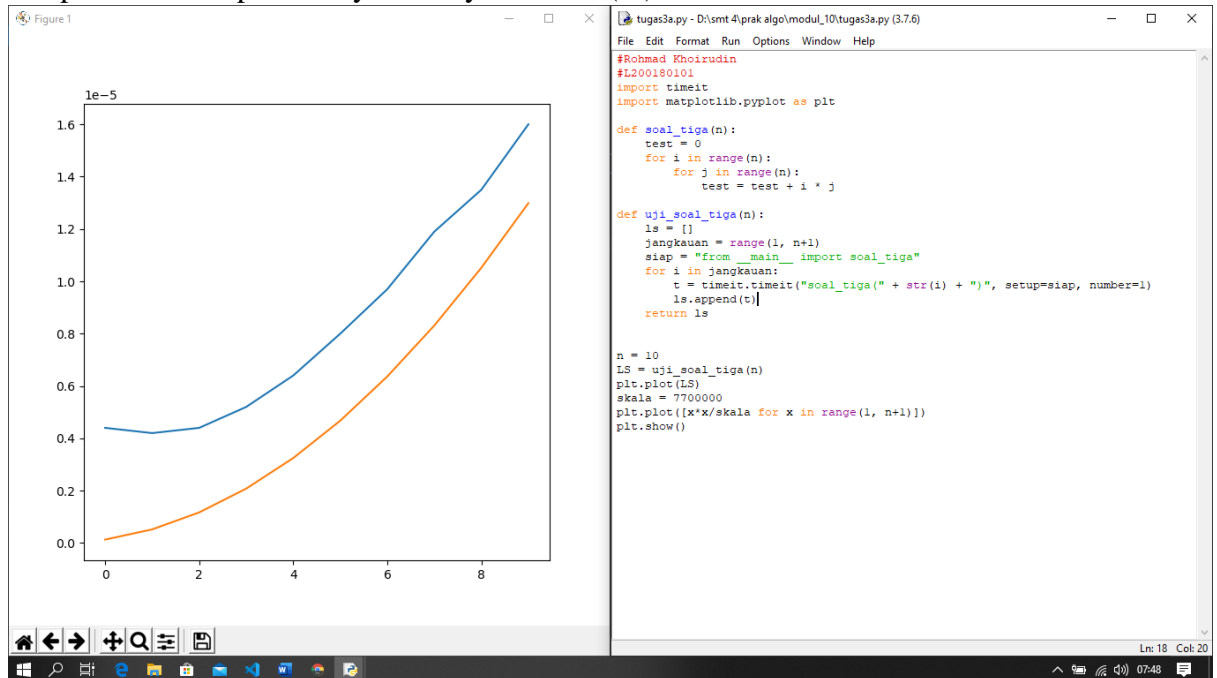
```
File Edit Selection View Go Run Terminal Help tugas2.py - modul_10 - Visual Studio Code
tugas3d.py • tugas3e.py • tugas3f.py • tugas3g.py • Settings • tugas7.py • tugas8.py • tugas3b.py • tugas3c.py • tugas2.py x
tugas2.py > ...
27 while i < len1:
28     arr[i] = left[i]
29     k += 1
30     i += 1
31
32 while j < len2:
33     arr[k] = right[j]
34     k += 1
35     j += 1
36
37 # timsort
38 def timsort(arr, n):
39     for i in range(0, n, n//64):
40         insertionSort(arr, i, min((i+31), (n-1)))
41
42     size = n//64
43     while size < n:
44         for left in range(0, n, 2*size):
45             mid = left + size - 1
46             right = min((left + 2*size - 1), (n-1))
47             merge(arr, left, mid, right)
48             size = 2*size
49
50 g = [5, 21, 7, 23, 19]
51 n = len(g)
52 print("g = ", g)
53 z = [5, 21, 7, 23, 19]
54 print("z = ", z)
55 a = [0, 5, 4, 6, 8, 1, 2]
56 print("a = ", a)
57
58 # timsort
59
```

```
File Edit Selection View Go Run Terminal Help tugas2.py - modul_10 - Visual Studio Code
tugas3d.py • tugas3e.py • tugas3f.py • tugas3g.py • Settings • tugas7.py • tugas8.py • tugas3b.py • tugas3c.py • tugas2.py x
tugas2.py > ...
64 a = [9, 5, 4, 6, 8, 1, 2]
65 print("a = ", a)
66
67 # timsort
68 timsort(g, len(g))
69 print("=====timsort")
70 print("g = ", g)
71 # a.sort()
72
73 print("===== a.sort()")
74 a.sort()
75 print("a = ", a)
76
77 # Sorted
78 z = [5, 21, 7, 23, 19]
79 print("=====sorted")
80 sorted(z)
81 print("z = ", z)
82
83 # bestcase Sorted
84 print("=====Best Case Sorted=====")
85
86
87 def bestcase():
88     for i in range(5):
89         L = list(range(3000))
90         awal = time.time()
91         U = sorted(L)
92         akhir = time.time()
93         print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
94
95 bestcase()
96
97
98 # Average Case
99 print("=====Average Case Sorted=====")
100 def averagecase():
101     for i in range(5):
102         L = list(range(3000))
103         random.shuffle(L)
104         awal = time.time()
105         U = sorted(L)
106         akhir = time.time()
107         print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
108
109 averagecase()
110
111 # worst case
112 print("=====Worst Case Sorted=====")
113 def worstcase():
114     for i in range(5):
115         L = list(range(3000))
116         L.reverse()
117         awal = time.time()
118         U = sorted(L)
119         akhir = time.time()
120         print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
121
122 worstcase()
123
```

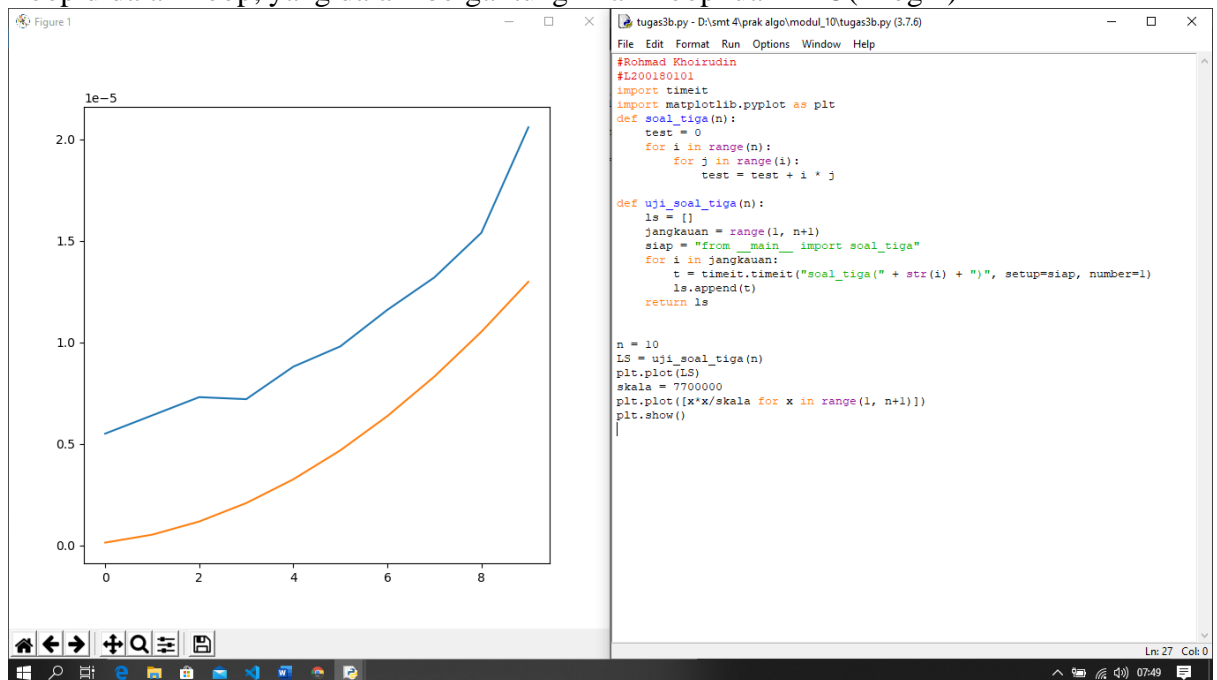
```
File Edit Selection View Go Run Terminal Help tugas2.py - modul_10 - Visual Studio Code
tugas3d.py • tugas3e.py • tugas3f.py • tugas3g.py • Settings • tugas7.py • tugas8.py • tugas3b.py • tugas3c.py • tugas2.py x
tugas2.py > ...
91 U = sorted(L)
92 akhir = time.time()
93 print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
94
95 bestcase()
96
97 # Average Case
98 print("=====Average Case Sorted=====")
99 def averagecase():
100     for i in range(5):
101         L = list(range(3000))
102         random.shuffle(L)
103         awal = time.time()
104         U = sorted(L)
105         akhir = time.time()
106         print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
107
108 averagecase()
109
110 # worst case
111 print("=====Worst Case Sorted=====")
112 def worstcase():
113     for i in range(5):
114         L = list(range(3000))
115         L.reverse()
116         awal = time.time()
117         U = sorted(L)
118         akhir = time.time()
119         print("mengurutkan %d bilangan, memerlukan waktu %8.7f detik" % (len(L), akhir - awal))
120
121 worstcase()
122
```

3) Tentukan running time-nya, $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$, atau $O(n^3)$

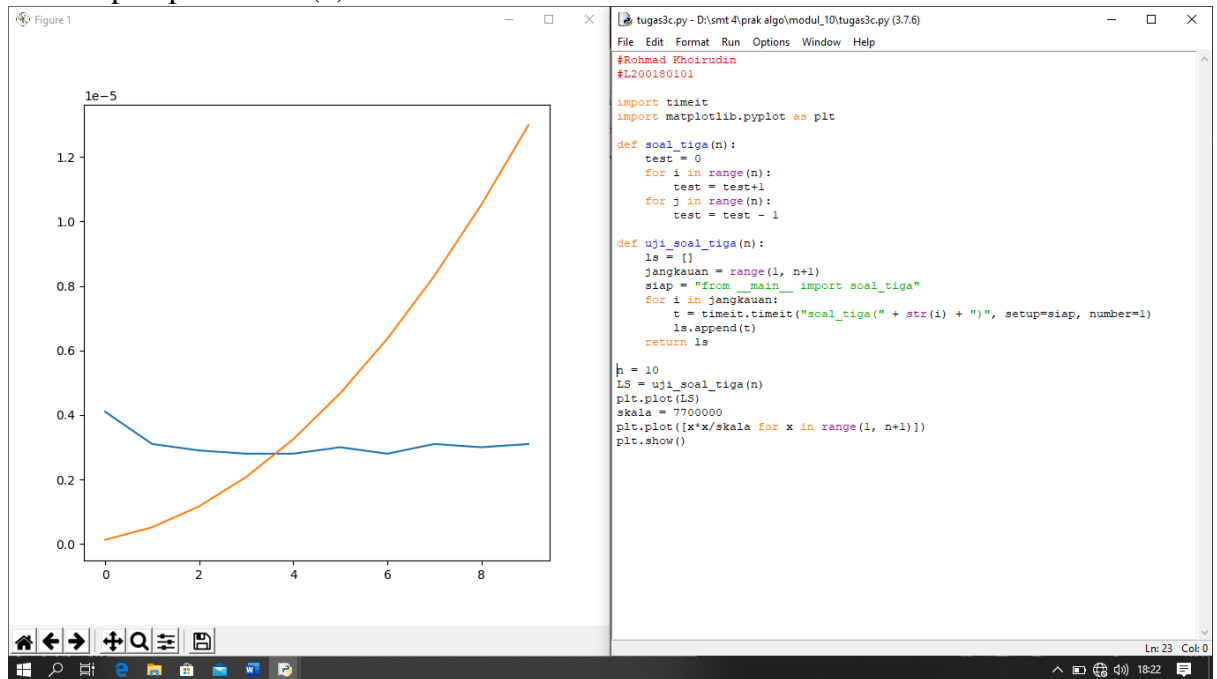
- Loop didalam loop keduanya sebanyak $n \rightarrow O(n^2)$



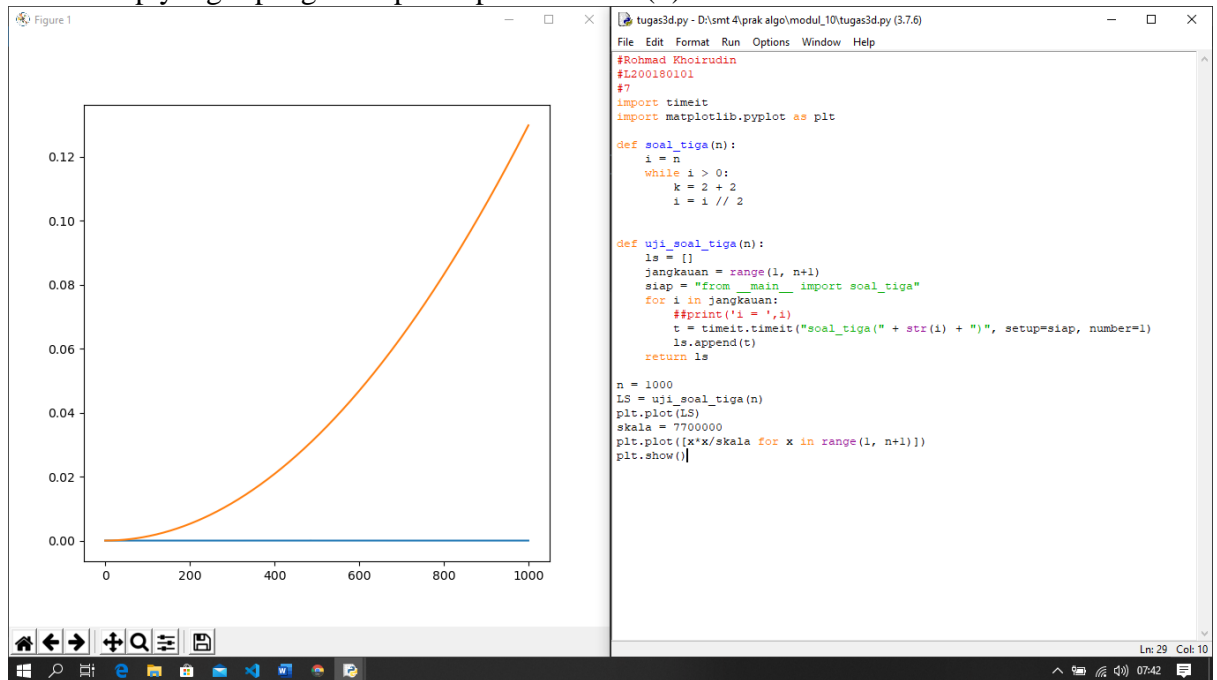
- Loop didalam loop, yang dalam bergantung nilai i loop luar $\rightarrow O(n \log n)$



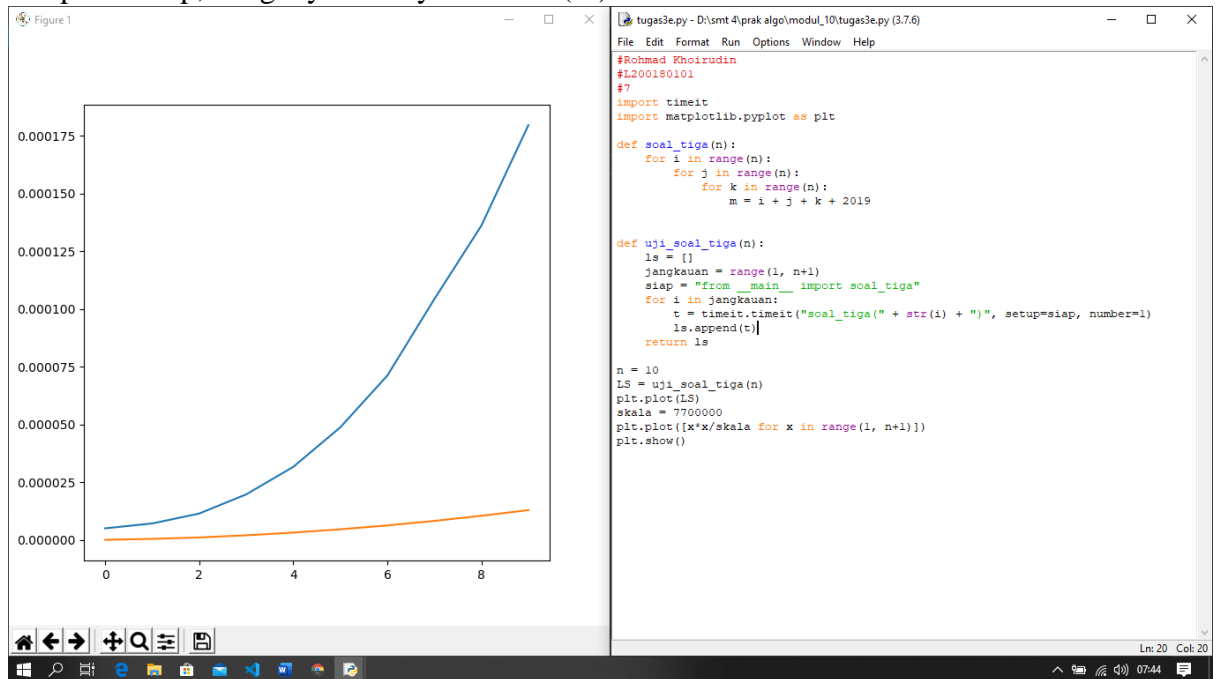
- Dua loop terpisah $\rightarrow O(1)$



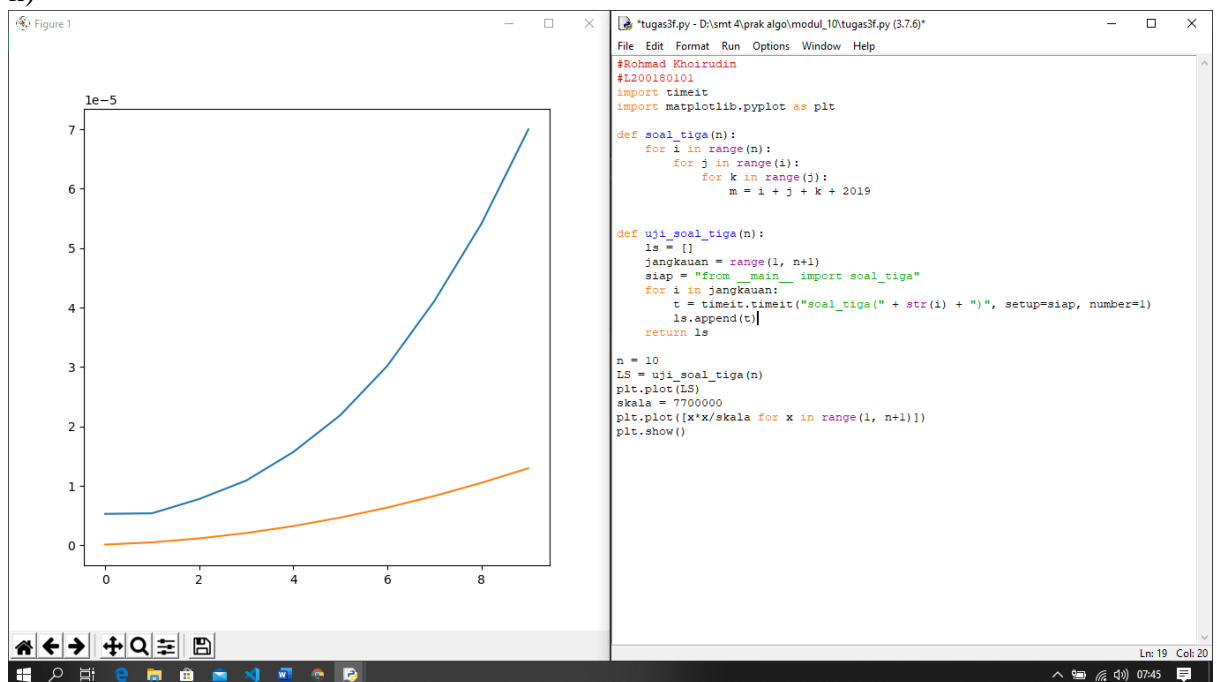
- While loop yang dipangkas separuh putaran $\rightarrow O(1)$



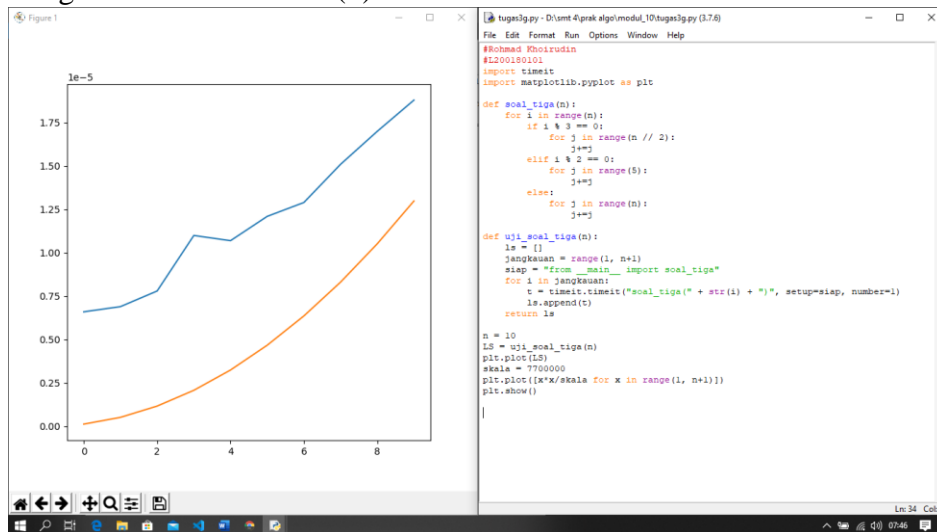
- Loop in a loop, ketiganya sebanyak $n \rightarrow O(n^3)$



- Loop in a loop, dengan loop dalam sebanyak nilai loop luar terdekat $\rightarrow O(n \log n)$



- Fungsi ini termasuk $\rightarrow O(n)$



- 4) Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat
 $n \log_2 n$ 4^n $10 \log_2 n$ $5n^2$ $\log_4 n$ $12n^6$ $2^{\log_2 n}$ n^3 !

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

- 5) Tentukan $O(\cdot)$ dari fungsi-fungsi berikut ini.

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

- 6) Carilah di internet kompleksitas metode – metode object list di python.

- Google python list methods complexity
- Kunjungi <https://wiki.python.org/moin/TimeComplexity>

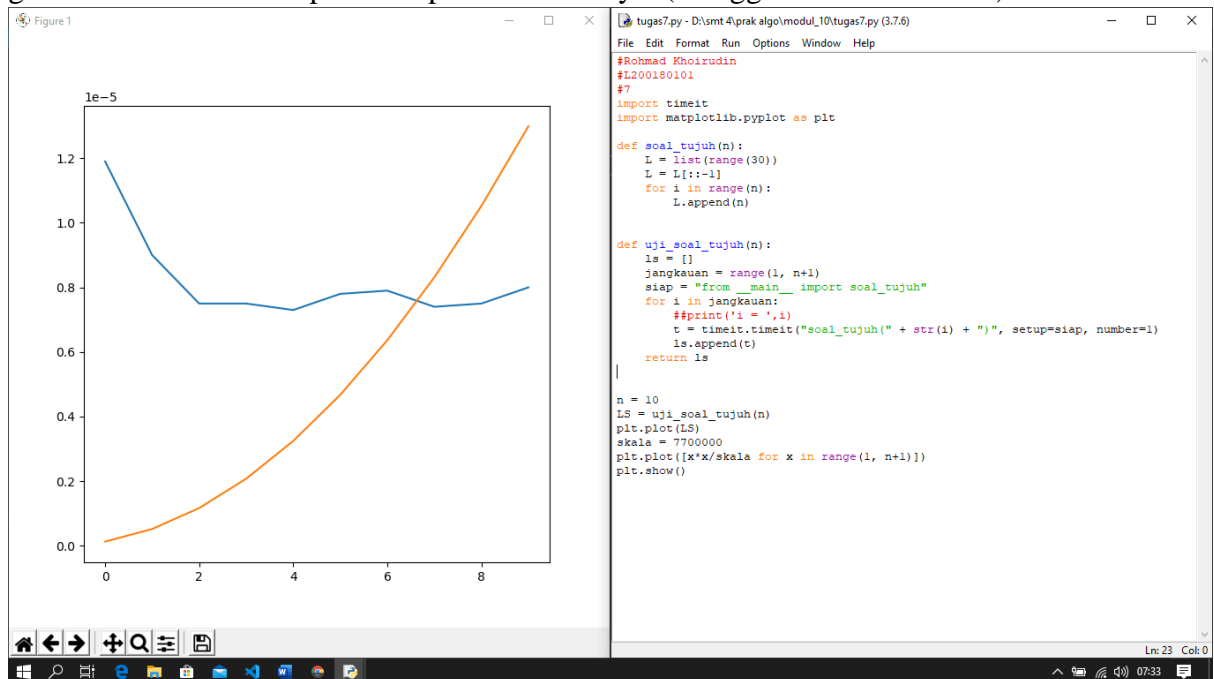
list

The Average Case assumes parameters generated uniformly at random.

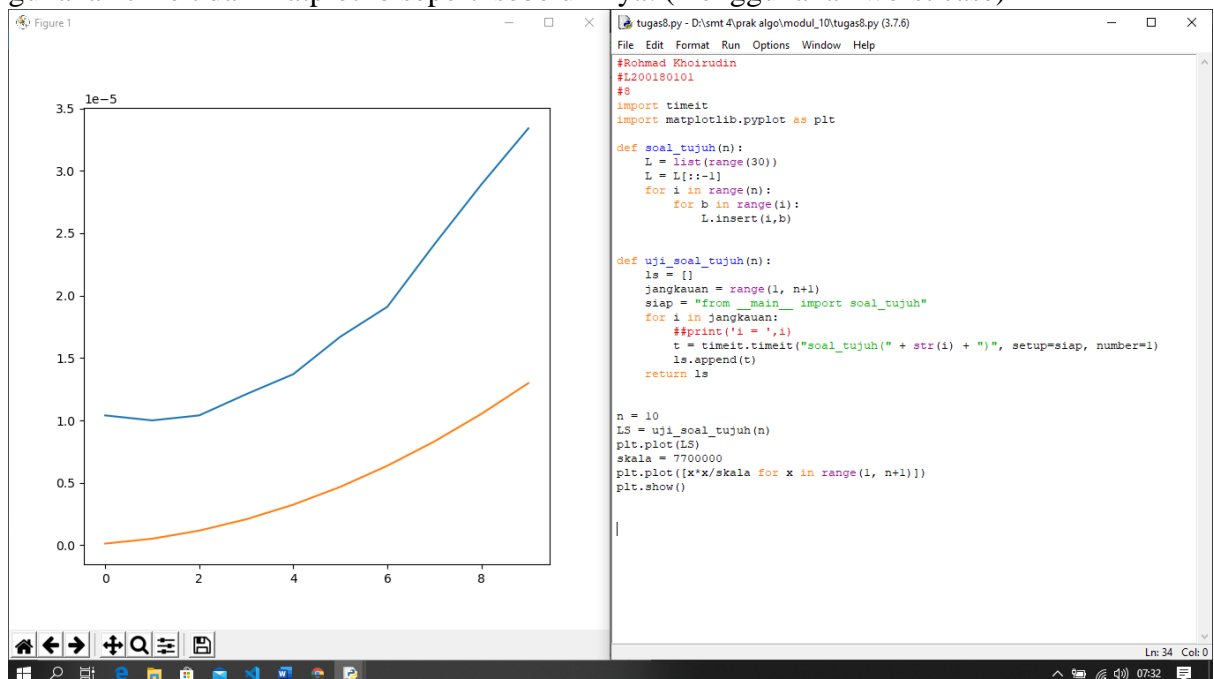
Internally, a list is represented as an array; the largest costs come from growing beyond the current allocation size (because everything must move), or from inserting or deleting somewhere near the beginning (because everything after that must move). If you need to add/remove at both ends, consider 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	$O(k)$	$O(k)$
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	$O(n)$	$O(n)$
min(s), max(s)	$O(n)$	$O(n)$
Get Length	$O(1)$	$O(1)$

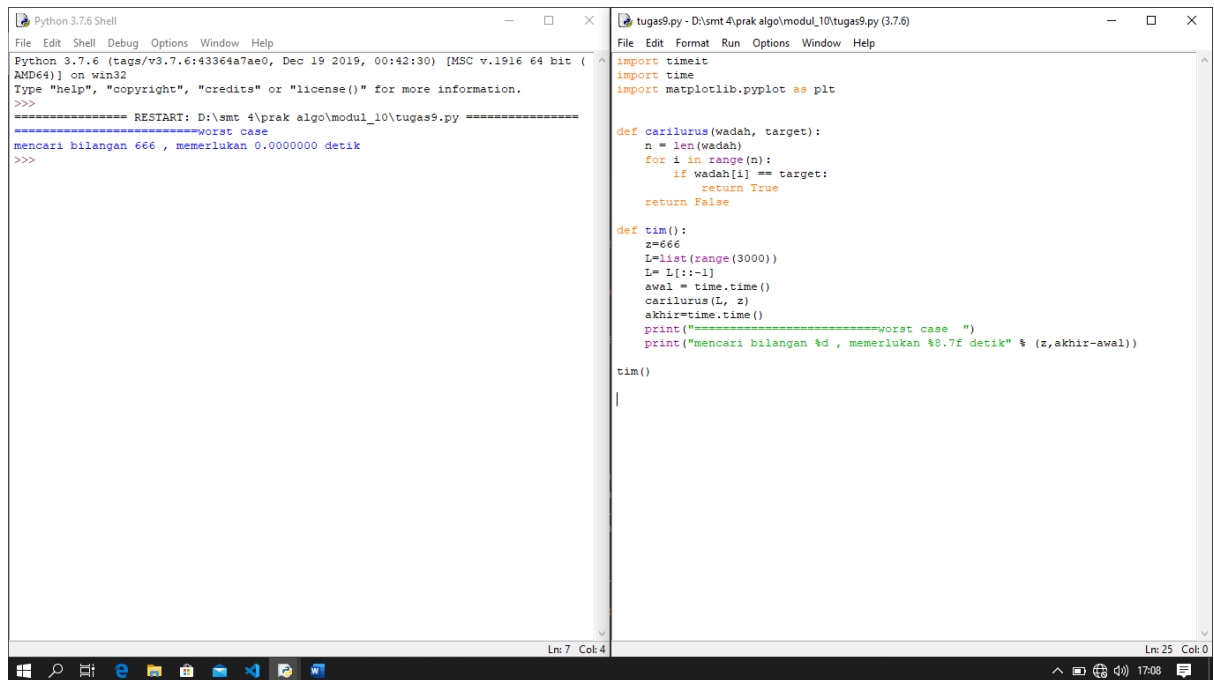
- 7) Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode `append()` adalah $O(1)$. gunakan `timeit` dan `matplotlib` seperti sebelumnya. (menggunakan worst case)



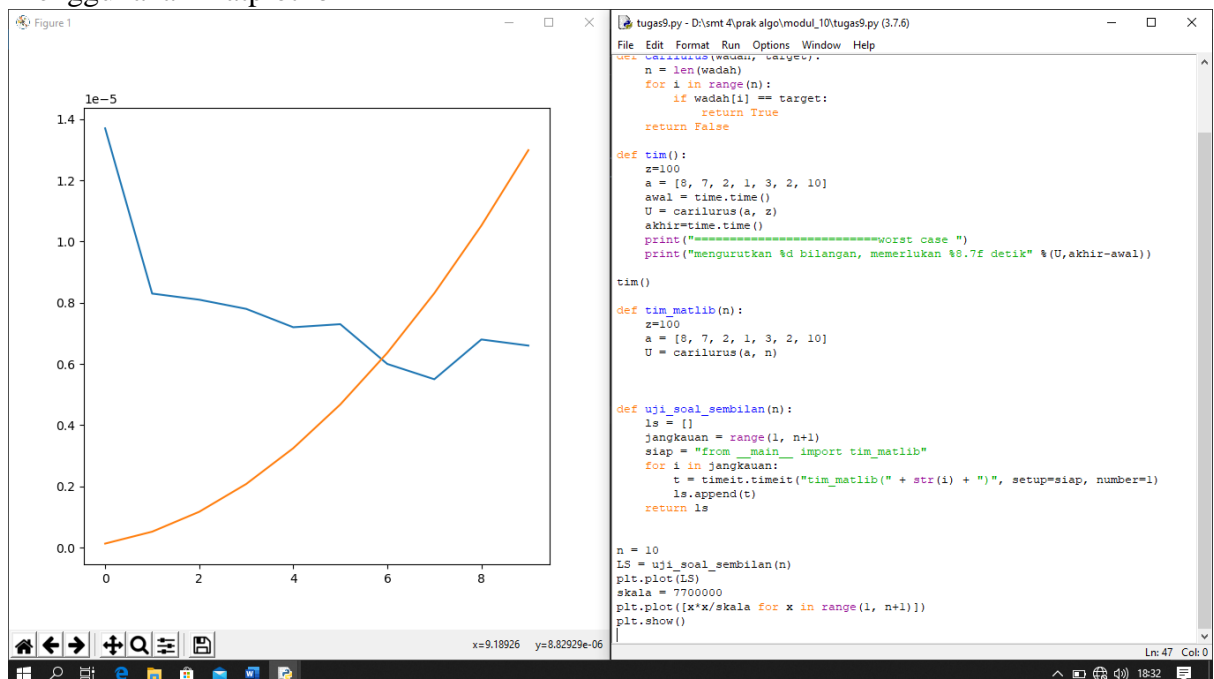
- 8) Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode `insert()` adalah $O(n)$. gunakan `timeit` dan `matplotlib` seperti sebelumnya. (menggunakan worst case)



- 9) Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah-suatu-nilai-berada-di-suatu-list mempunyai kompleksitas $O(n)$. Gunakan `timeit` dan `matplotlib`. Menggunakan `timeit` worstcase



Menggunakan matplotlib



10) Carilah di internet kompleksitas metode metode pada object dict di python dict

The Average Case times listed for dict objects assume that the hash function for the objects is sufficiently robust to make collisions uncommon. The Average Case assumes the keys used in parameters are selected uniformly at random from the set of all keys.

Note that there is a fast-path for dicts that (in practice) only deal with str keys; this doesn't affect the algorithmic complexity, but it can significantly affect the constant factors: how quickly a typical program finishes.

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 $O(\cdot)$, ada pula notasi big-Theta $\Theta(\cdot)$ dan notasi big-omega $\Omega(\cdot)$ apakah beda diantara ketiganya ?

Big-O

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

$$f(n) = n^2 - n < n^2 = g(n).$$

Big-omega

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

$$f = \Omega(g) \text{ if } g = O(f)$$

Big-Theta

menjelaskan batasan ketat dari suatu algoritma, batasnya dari atas dan bawah. Dengan menggunakan contoh Penyisipan akan diketahui bahwa laju pertumbuhan paling banyak adalah $O(n^2)$ dan setidaknya $\Omega(n)$.

$$f = \Theta(g) \text{ if } f = O(g) \text{ and } f = \Omega(g)$$

- 12) Apa yang dimaksud dengan amortized analysis dalam analisis algoritma?

⇒ Amortized Analysis digunakan untuk algoritma di mana operasi sesekali sangat lambat, tetapi sebagian besar operasi lainnya lebih cepat. Dalam Amortized Analysis, menganalisis urutan operasi dan menjamin waktu rata-rata terburuk yang lebih rendah dari waktu terburuk dari operasi mahal tertentu.