LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA
MODUL 10
”ANALISIS ALGORITMA”

Oleh:

NAMA : Daffa Putra Alwansyah
NIM : L200190031
KELAS : B
PRODI : Informatika

Fakultas Komunikasi dan Informatika
Universitas Muhammadiyah Surakarta

LATIHAN

10.1 Sebuah Contoh
################################

```
import time

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

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

print("cara 2",jumlahkan_cara_2(1000000))

for i in range(5):
    awal = time.time()
    h = jumlahkan_cara_2(1000000)
    akhir = time.time()
    print("Jumlah adalah %d,memerlukan waktu %9.8f detik" % (h,akhir-awal))
```

```
= RESTART: D:\Kuliah\Semester 4\Praktikum Algoritma dan Strukt
atihan_Modul10.py
cara 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
>>>
```

10.3 Kasus terburuk,rata-rata dan terbaik
###

```
import time
import random

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====================")

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))

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))

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))
```

```
= RESTART: D:\Kuliah\Semester 4\Praktikum Algoritma dan Struktur
atihan_Modul10.py
==============================Average Case==================
mengurutkan 3000 bilangan,memerlukan waktu 0.9730554 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0310593 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0940626 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0150583 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.0110576 detik
==============================Worst Case==================
mengurutkan 3000 bilangan,memerlukan waktu 1.7511001 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.9051089 detik
mengurutkan 3000 bilangan,memerlukan waktu 2.0011146 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.8041034 detik
mengurutkan 3000 bilangan,memerlukan waktu 1.9031086 detik
==============================Best Case==================
mengurutkan 3000 bilangan,memerlukan waktu 0.0009999 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0020001 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0009999 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0009999 detik
mengurutkan 3000 bilangan,memerlukan waktu 0.0010002 detik
>>>
```

```
###### 10.4 Menganalisis Kode Python ######
##########################################

x = 5
y = x
z = x + y*8
d = x > 0 and x < 100
f = [3,2,4,5]
v = f[0:2]

i = 32
```

```
while i >= i:
    count += 1
    i = 1//2
    print('i =',i,' ','count =', count)
```

```
>>> x = 5
>>> y = x
>>> z = x + y*8
>>> d = x > 0 and x<100
>>> f = [3,2,4,5]
>>> v = f[0:2]
>>> print['x=',x]
Traceback (most recent call last):
  File "<pyshell#44>", line 1, in <module>
    print['x=',x]
TypeError: 'builtin_function_or_method' object is
>>> print('x=',x)
x= 5
>>> print('z=',z)
z= 45
>>> print('d=',d)
d= True
>>> print('v=',v)
v= [3, 2]
>>> print('f=',f)
f= [3, 2, 4, 5]
>>> count=0
>>> i=32
>>> while i >= i:
    count += 1
    i = 1//2
    print('i =',i,' ','count =', count)
```

```
i = 0    count = 499
i = 0    count = 500
i = 0    count = Traceback (most recent call last):
  File "<pyshell#54>", line 4, in <module>
    print('i =',i,' ','count =', count)
KeyboardInterrupt
>>> print(count)
501
>>>
```

10.5 Analisis pewaktuan menggunakan timeit
##

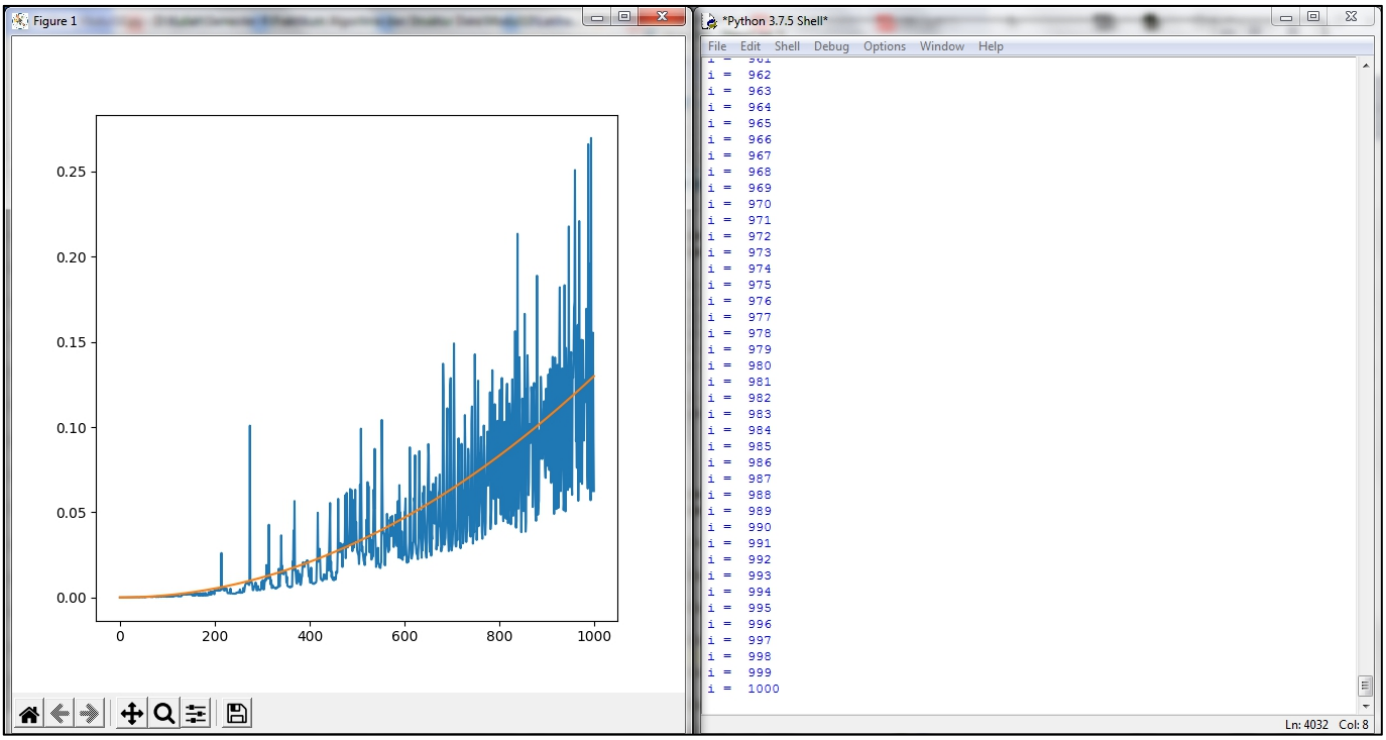
```
from timeit import timeit

timeit('sqrt(2)','from math import sqrt',number = 10000)

timeit('sqrt(2)','from math import sqrt',number = 100000)

timeit('sqrt(2)','from math import sqrt',number = 1000000)

timeit("1+2")

timeit("sin (pi/3)", setup = "from math import sin,pi")
```

timeit("sin (1.047)", setup = "from math import sin")

```
>>> from timeit import timeit
>>> timeit('sqrt(2)','from math import sqrt',number = 10000)
0.001599267999999654
>>> timeit('sqrt(2)','from math import sqrt',number = 100000)
0.012933273999999884
>>> timeit('sqrt(2)','from math import sqrt',number = 1000000)
0.28660258600000077
>>> timeit("1+2")
0.023732726999998732
>>> timeit("sin (pi/3)", setup = "from math import sin,pi")
0.1843159270000001
>>> timeit("sin (1.047)", setup = "from math import sin")
0.18882023799999814
>>>
```

#######10.5.1 Melihat O(n^2) pada nested loop#######
##

```
import timeit
import matplotlib.pyplot as plt

## Ini fungsi nested loop yang akan diuji:
def kalangBersusuh(n):
    for i in range(n):
        for j in range (n):
            i+j

## Ini fungsi pengujinya:
def ujiKalangBersusuh(n):
    ls=[]
    jangkauan=range(1,n+1)
    siap = "from __main__ import kalangBersusuh"
    for i in jangkauan:
        print('i = ',i) #baris ini bisa dihidupkan atau dimatikan
        t=timeit.timeit("kalangBersusuh(" +str(i) +")",setup=siap,number=1)
        ls.append(t)
    return ls

## Pemanggilan pengujian
n = 1000
LS = ujiKalangBersusuh(n)
## LS adalah list hasil uji kecepatan, dari n sedikit ke banyak.

## Menggambar grafik. dibawah ini saja yang diulang saat me-nyetel skala.
plt.plot(LS)    # Mem-plot hasil uji
skala=7700000   # <--------setel skala ini sesuai hasilmu
plt.plot([x*x/skala for x in range (1,n+1)]) # Grafik x^2 untuk pembanding
plt.show()      # Tunjukkan plotnya
```



Modul 10 Analisis Algoritma

1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul timeit, yakni

```
#==========No 1==========
from timeit 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):
    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))

print("=====Jumlahkan 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))
```

```
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))
    random.shuffle(L)
    t = timeit("insertionSort(L)", setup=siap, number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L), t))
```

```
= RESTART: D:/Kuliah/Semester 4/Praktikum Algoritma dan Struktur Data/Modul10/Mo
dul 10.py
=====Jumlahkan Cara 1=====
Jumlah adalah 500000500000, memerlukan 0.19963375 detik
Jumlah adalah 500000500000, memerlukan 0.06569945 detik
Jumlah adalah 500000500000, memerlukan 0.12770126 detik
Jumlah adalah 500000500000, memerlukan 0.11026396 detik
Jumlah adalah 500000500000, memerlukan 0.07995871 detik
=====Jumlahkan Cara 2=====
Jumlah adalah 500000500000, memerlukan 0.00002490 detik
Jumlah adalah 500000500000, memerlukan 0.00000146 detik
Jumlah adalah 500000500000, memerlukan 0.00000146 detik
Jumlah adalah 500000500000, memerlukan 0.00000146 detik
Jumlah adalah 500000500000, memerlukan 0.00000146 detik
=====Insertion Sort=====
Mengurutkan 3000 bilangan, memerlukan 1.0060840 detik
Mengurutkan 3000 bilangan, memerlukan 0.9908663 detik
Mengurutkan 3000 bilangan, memerlukan 1.0547436 detik
Mengurutkan 3000 bilangan, memerlukan 1.0358626 detik
Mengurutkan 3000 bilangan, memerlukan 1.1298333 detik
>>>
```

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**

```
#==========No 2==========
from timeit import timeit
import random

print("=====Sorted average 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))
```

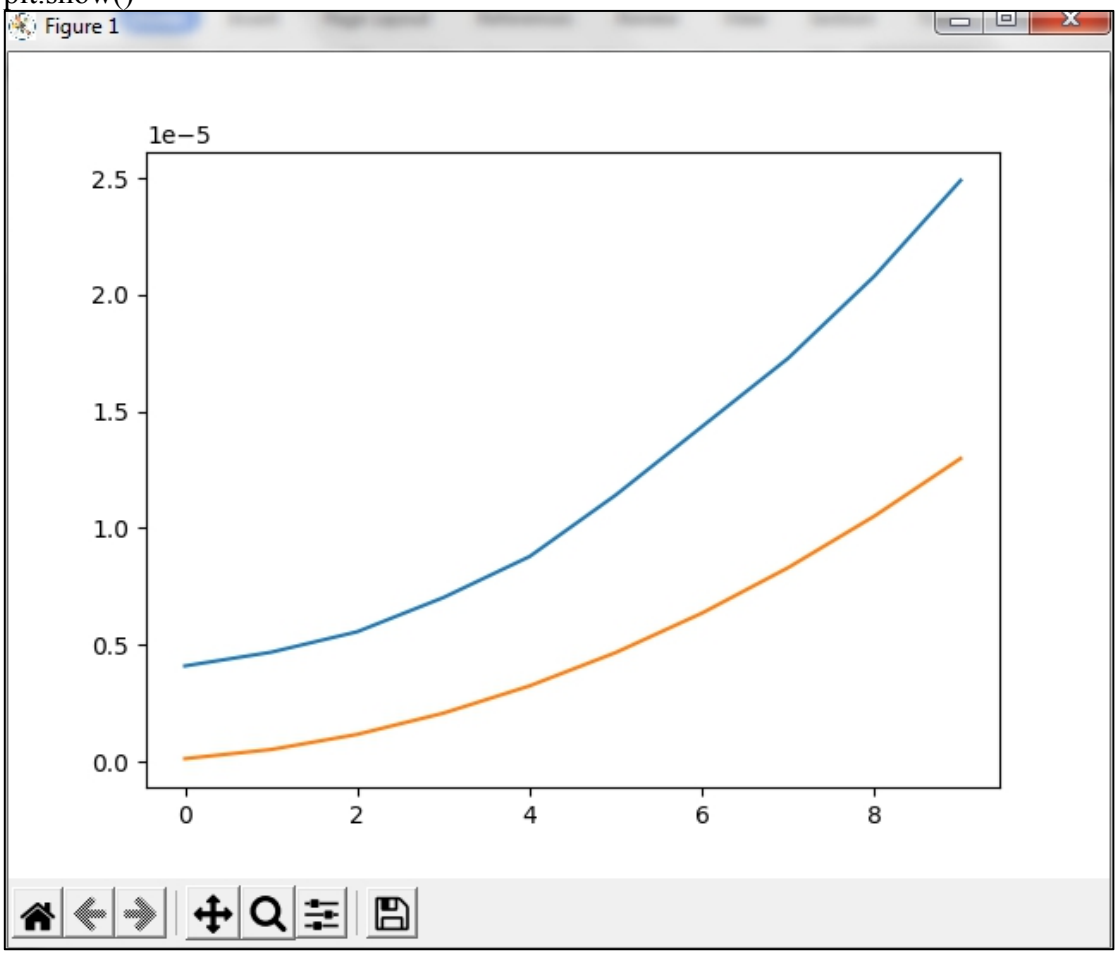
```
= RESTART: D:/Kuliah/Semester 4/Praktikum Algoritma dan Struktur Data/Modul10/Mo
dul 10.py
=====Sorted average case=====
Mengurutkan 3000 bilangan, memerlukan 0.0005193 detik
Mengurutkan 3000 bilangan, memerlukan 0.0005126 detik
Mengurutkan 3000 bilangan, memerlukan 0.0005337 detik
Mengurutkan 3000 bilangan, memerlukan 0.0005436 detik
Mengurutkan 3000 bilangan, memerlukan 0.0327501 detik
=====Sorted best case=====
Mengurutkan 3000 bilangan, memerlukan 0.0000630 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000557 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000615 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000636 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000618 detik
=====Sorted worst case=====
Mengurutkan 3000 bilangan, memerlukan 0.0000609 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000641 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000706 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000615 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000612 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(n2) atau O(n3) atau yang lain. Untuk memulai analisis, ambil suatu nilai n tertentu lalu ikuti apa yang terjadi di kode itu.

```
#==========No 3a==========
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    test = 0
```

```
    for i in range(n):
        for j in range(n):
            test = test + i * j

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 10
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



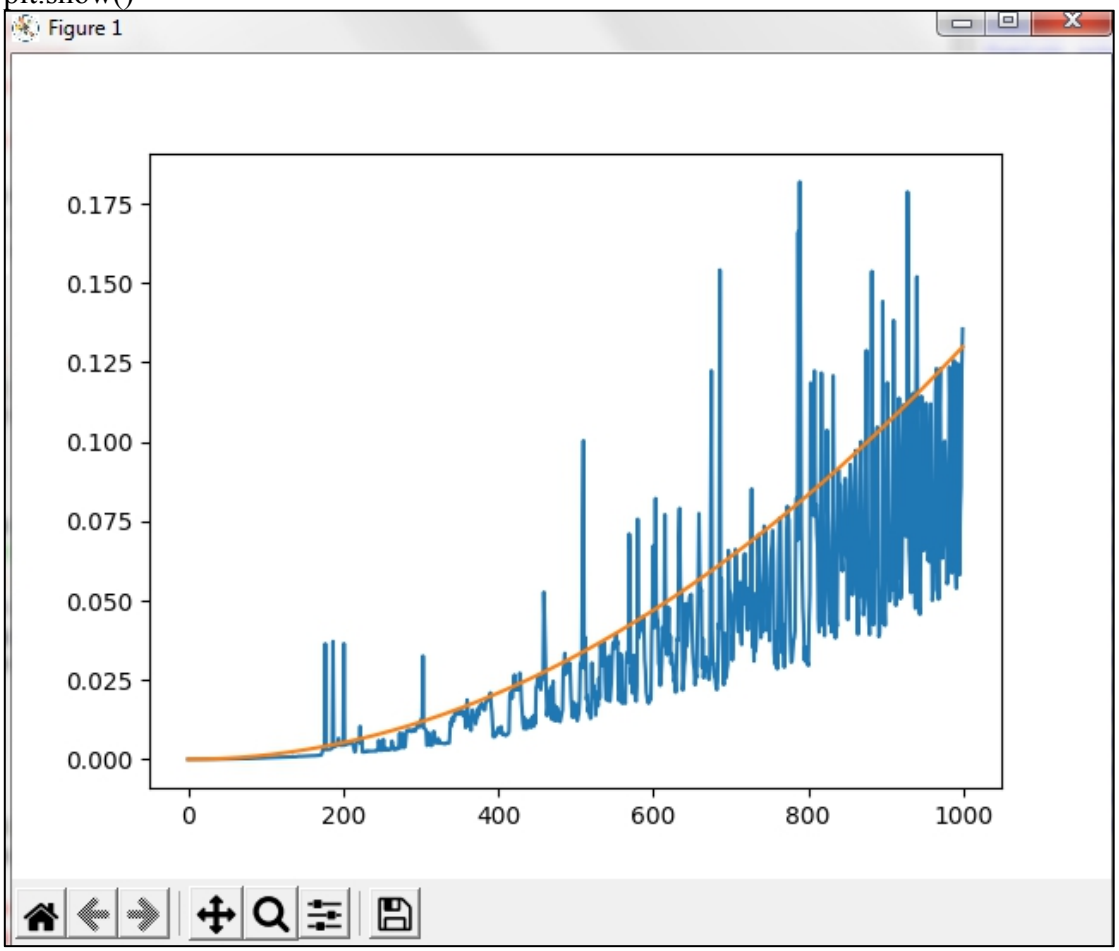
T(n) = c1+n*(n)
T(n) = c1+n^2
O(n) ≈ n^2

O(n^2)

#==========No 3b==========

```
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    test = 0
    for i in range(n):
        for j in range(i):
            test = test + i * j

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 1000
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
```

plt.show()

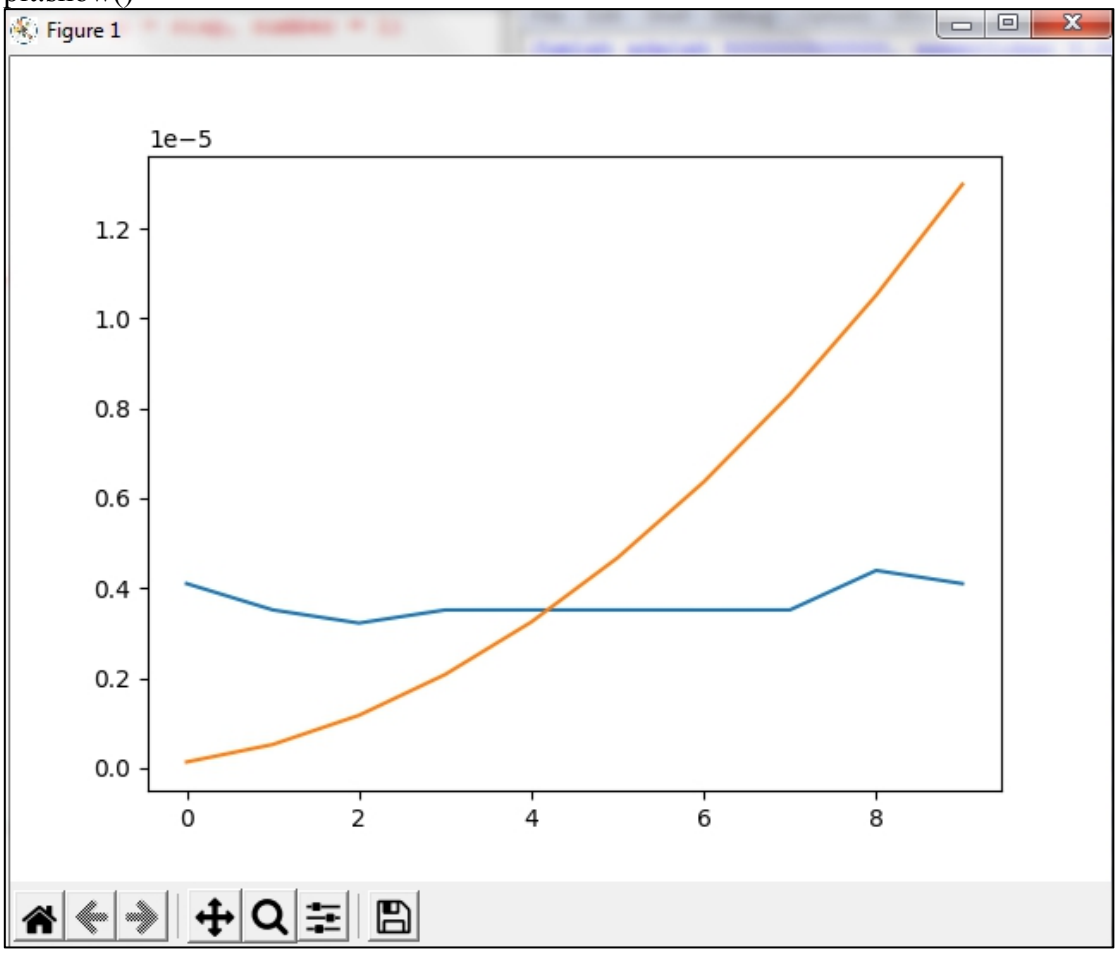


T(n) = c1+log n
T(n) = c1+log n
O(n) ≈ log n
O(log n)

#==========No 3c==========

```
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    test = 0
    for i in range(n):
        test = test + 1
    for j in range(n):
        test = test - 1

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
```

```
    return ls

n = 10
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```

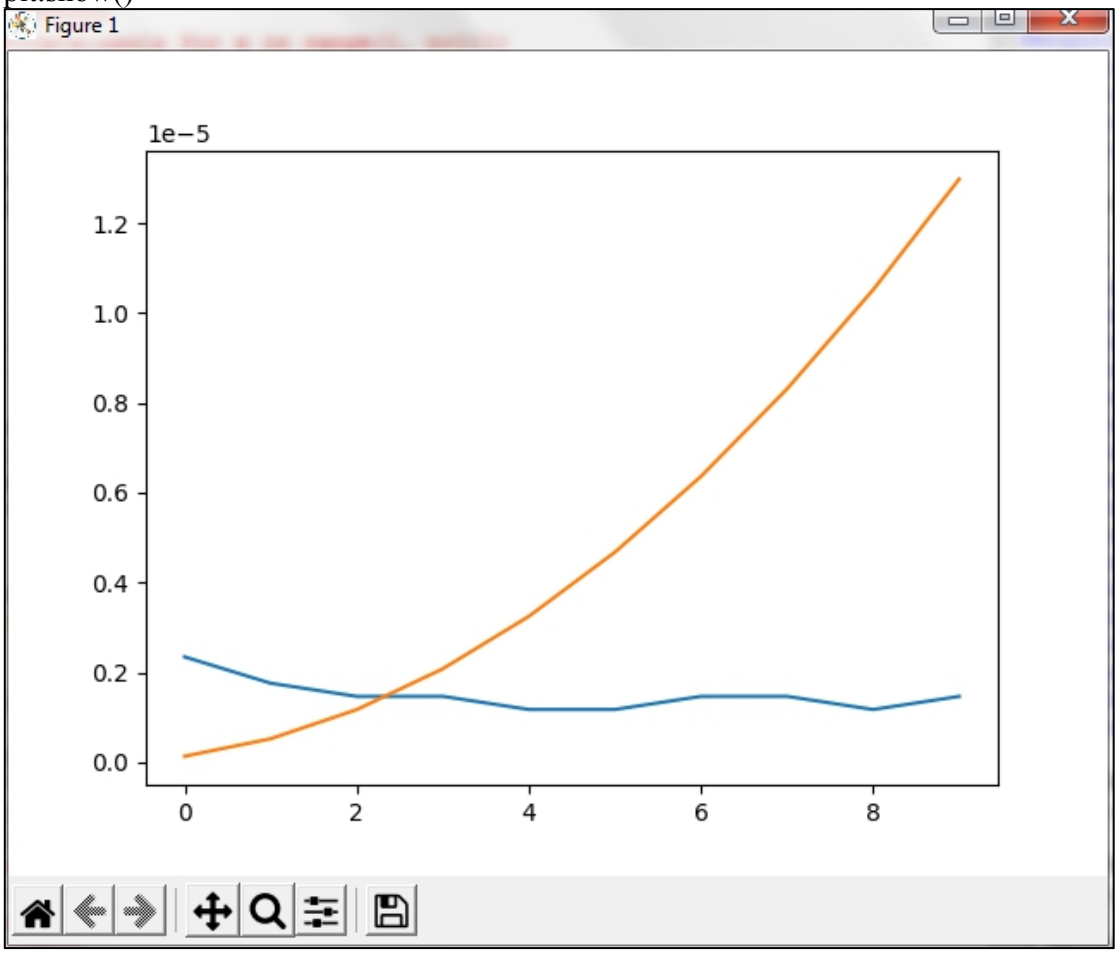


T(n) = c1(n) + c2(n)
T(n) = n + n
O(n) ≈ n
O(n)

#==========No 3d==========

```
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    i = n
    while i > 0:
```

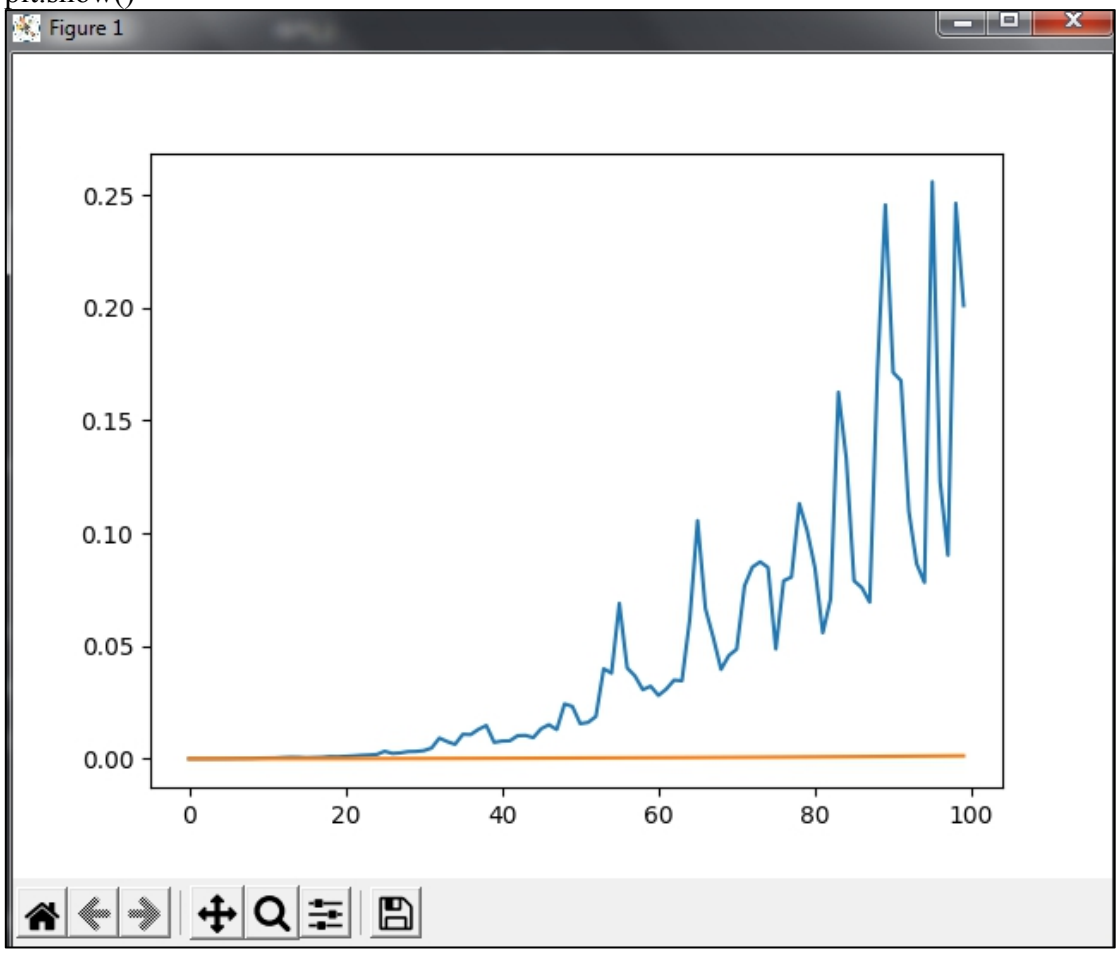
```
        k = 2 + 2
        i = i // 2

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 10
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



T(n) = c1(1) + c2(1)
O(n) ≈ 1
O(1)

```
#==========No 3e==========
import time
import random
```

```
import timeit
import matplotlib.pyplot as plt

def a(n):
    for i in range(n):
        for j in range(n):
            for k in range(n):
                m = i + j + k + 2019
def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 100
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```

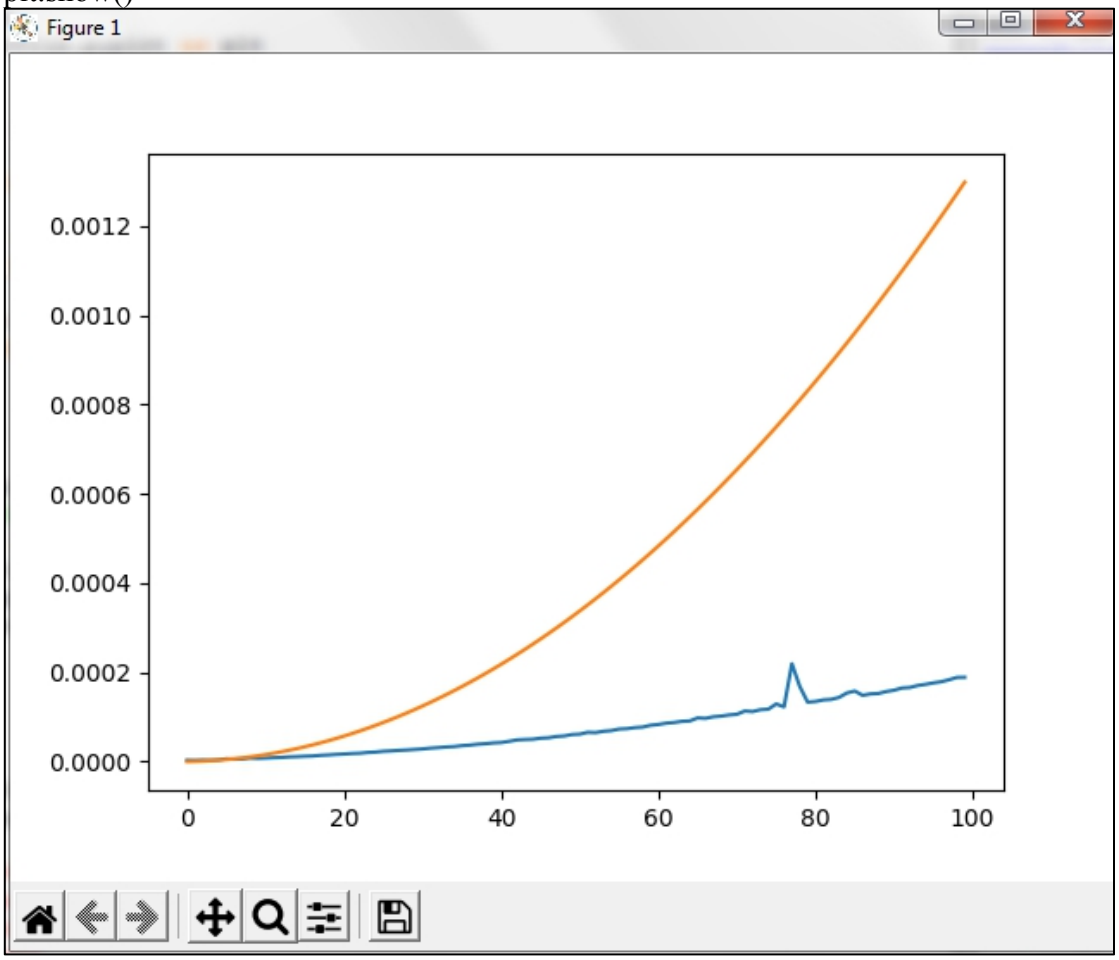


T(n) = n*(n*(n))

T(n) = n^3
O(n) ≈ n^3
O(n^3)

```
#==========No 3f==========
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    for i in range(n):
        if i % 3 == 0:
            for j in range(n // 2):
                j+=j
        elif i % 2 == 0:
            for j in range(5):
                j+=j
        else:
            for j in range(n):
                j+=j

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 100
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
```

plt.show()

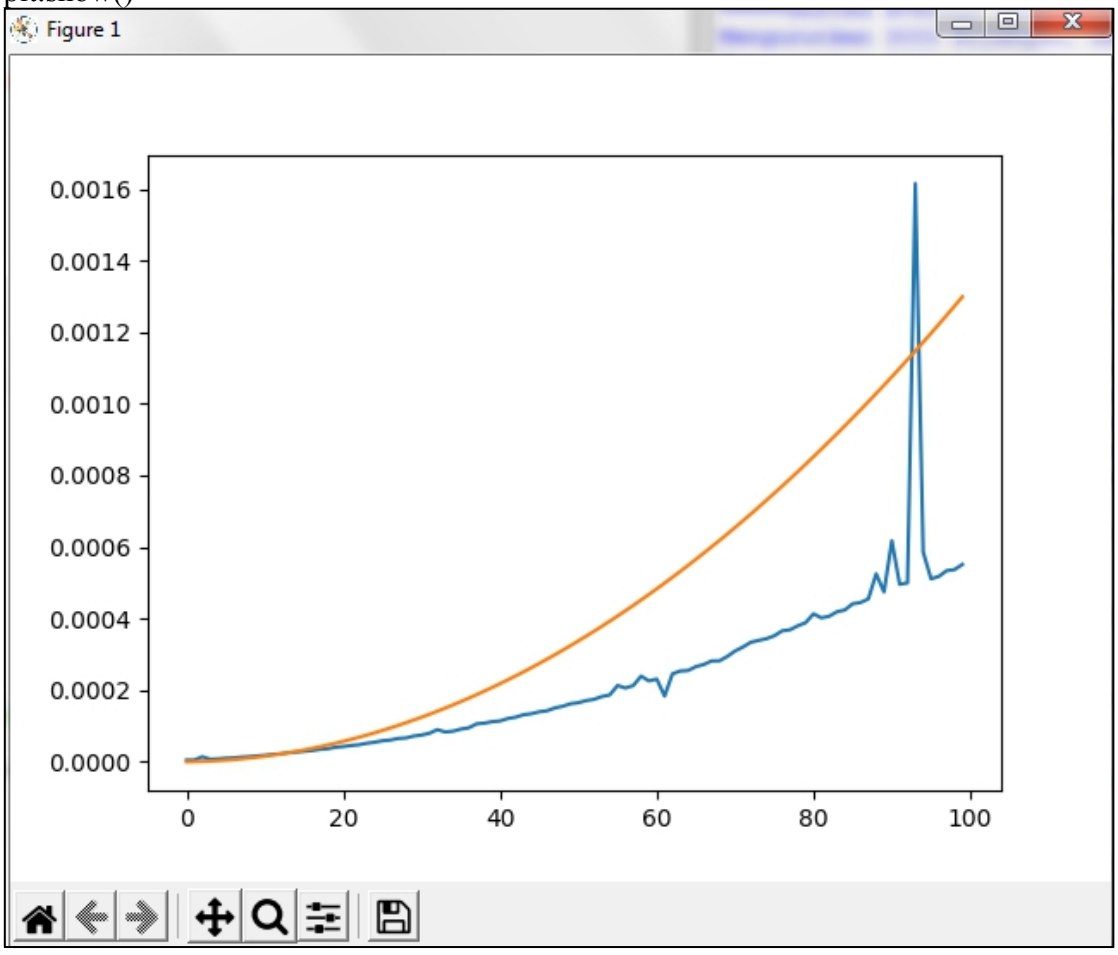


T(n) = c1(n) + c2(n) + c3(n)
T(n) = n + n + n
O(n) ≈ n
O(n)

#==========No 3g==========

```
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    for i in range(n):
        if i % 3 == 0:
            for j in range(n // 2):
                j+=j
        elif i % 2 == 0:
            for j in range(5):
                j+=j
        else:
            for j in range(n):
                j+=j
```

```
def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 100
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



O(n log n)

4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat:

log4n < 10log2n < n log2n < 2 log2n < 5n2 < n3 < 12n6 < 4n

5. Tentukan O(.) dari fungi-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)

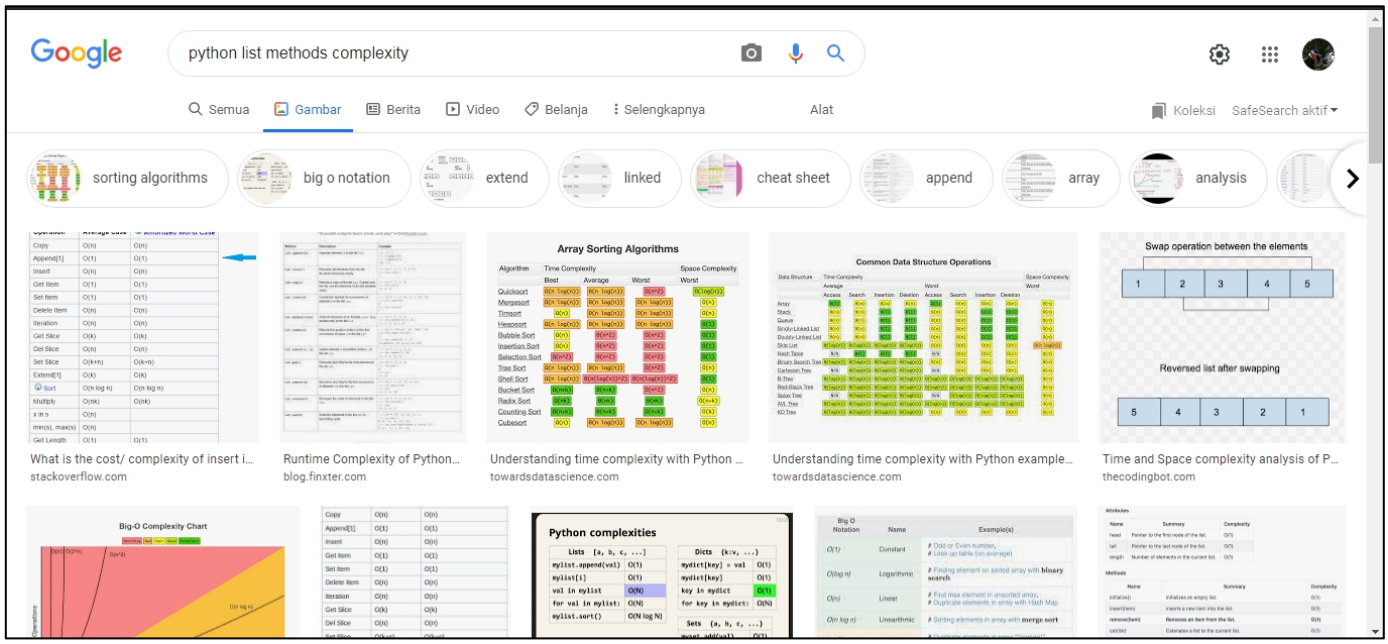
(f) T(n, k) = kn + log k = O(kn)

(g) T(n, k) = 8n + k log n + 800 = O(n)

(h) T(n, k) = 100kn + n = O(kn)

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

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



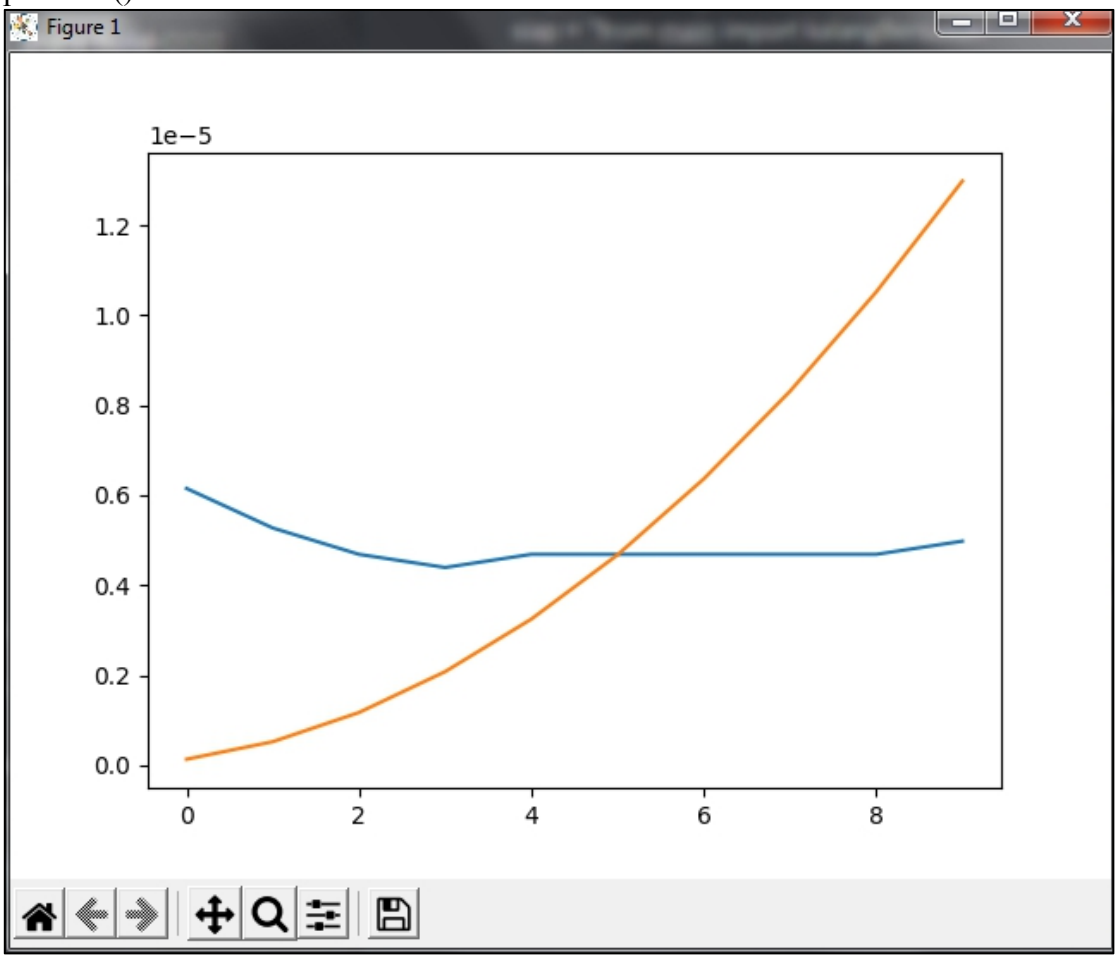
-Kunjungi https://wiki.python.org/moin/TimeComplexity

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[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)

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

```
###==========No 7==========
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    L = list(range(30))
    L = L[::-1]
    for i in range(n):
        L.append(n)

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
```

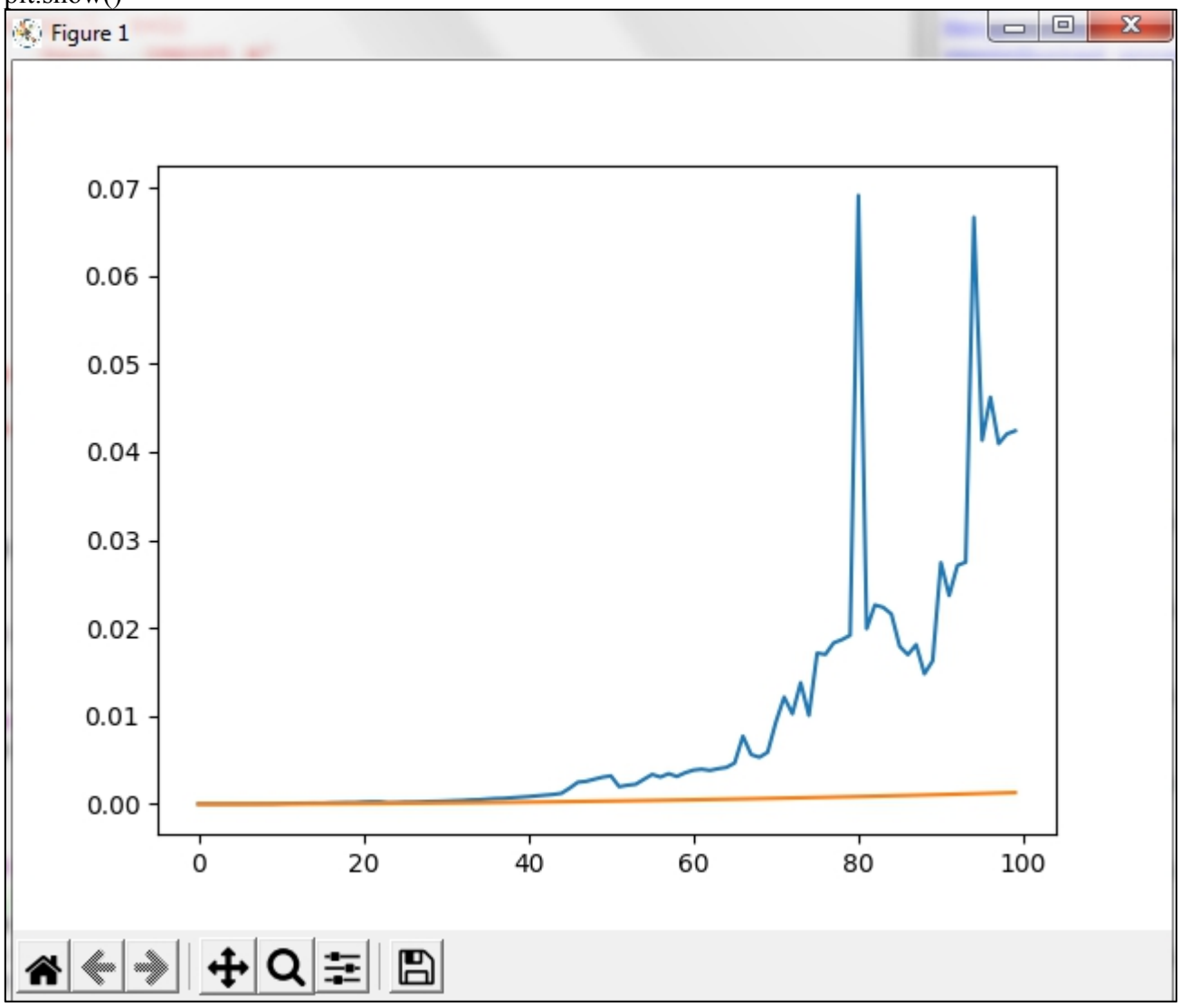
```
    return ls

n = 10
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



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

```
###==========No 8==========
import time
import random
import timeit
import matplotlib.pyplot as plt

def a(n):
    L = list(range(30))
    L = L[::-1]
    for i in range(n):
        for b in range(n):
            L.insert(i,b)

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 100
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
```

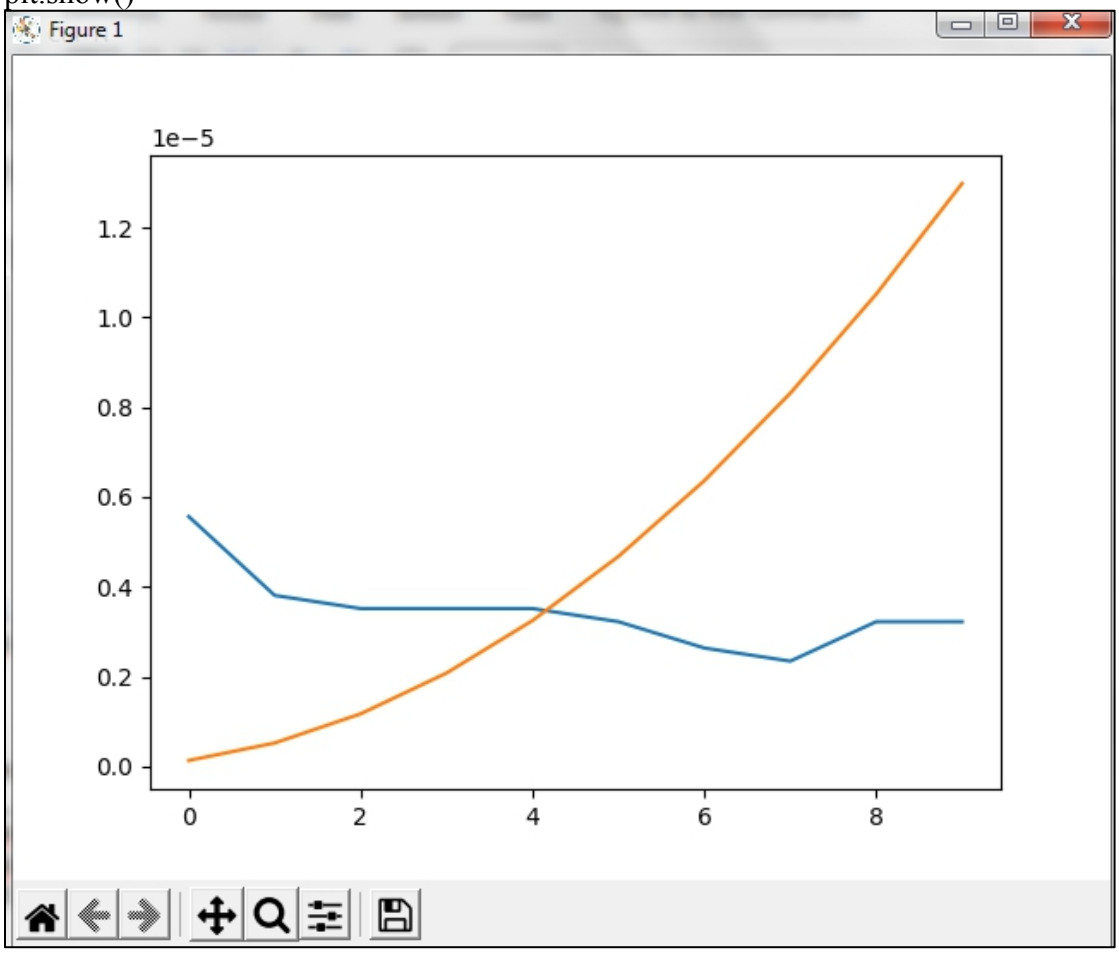
plt.show()



9. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah suatu nilai berada di suatu list mempunya kompleksitas O(n). Gunakan timeit dan matplotib seperti sebelumnya.

```
###==========No 9a==========
import timeit
import time
import matplotlib.pyplot as plt


def carilurus(wadah, target):
    n = len(wadah)
    for i in range(n):
        if wadah[i] == target:
            return True
    return False

def tim():
    z=100
    a = [8, 7, 2, 1, 3, 2, 10]
    awal = time.time()
    U = carilurus(a, z)
    akhir=time.time()
    print("Worst case")
    print("mengurutkan %d bilangan, memerlukan %8.7f detik" %(U,akhir-awal))

tim()
###==========No 9b==========
import time
import random
import timeit
import matplotlib.pyplot as plt

def carilurus(wadah, target):
    n = len(wadah)
    for i in range(n):
        if wadah[i] == target:
            return True
    return False

def tim():
    z=100
    a = [8, 7, 2, 1, 3, 2, 10]
    awal = time.time()
    U = carilurus(a, z)
    akhir=time.time()
    print("Worst case")
    print("mengurutkan %d bilangan, memerlukan %8.7f detik" %(U,akhir-awal))

tim()
```

```
def a(n):
    z=100
    a = [8, 7, 2, 1, 3, 2, 10]
    U = carilurus(a, n)

def ujia(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import a'
    for i in jangkauan:
        t = timeit.timeit('a(' + str(i) + ')', setup = siap, number = 1)
        ls.append(t)
    return ls

n = 10
LS = ujia(n)

plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



```
= RESTART: D:/Kuliah/Semester 4/Praktikum Algoritma
dul 10.py
Worst case
mengurutkan 0 bilangan, memerlukan 0.0000000 detik
>>>
```

10. Carilah di internet, kompelsitas 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 Θ(.) dan notasi big-Omega Ω (.) Apakah beda diantara ketiganya?

==> 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.

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 unyuk melakukan satu urutan operasi pada struktur data terhadap keseluruhan operasi yang dilakukan