PRAKTIKUM

ALGORITMA DAN STRUKTUR DATA

MODUL 10

Nama : NICKY JULYATRIKA SARI

NIM : L200200101

E

PROGRAM STUDI

INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

TAHUN 2021/2022

1. Kerjakan ulang contoh dan latihan di modul ini menggunakan modul timeit, yakni
   a. jumlahkan_cara_1

```
#Nicky Julyatrika sari l200200101
#modul 10
#nomer 1a


import time
from timeit import timeit
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 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC
v.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more inf
ormation.
>>>
========== RESTART: D:/MATKUL SMT 4/Praktikum ASD/modul 10/nom
er1.py ==========
Jumlah adalah 500000500000, memerlukan 0.10841340 detik
Jumlah adalah 500000500000, memerlukan 0.11113910 detik
Jumlah adalah 500000500000, memerlukan 0.12639520 detik
Jumlah adalah 500000500000, memerlukan 0.14049590 detik
Jumlah adalah 500000500000, memerlukan 0.12683530 detik
>>>
```

   b. jumlahkan_cara_2

```
#Nicky Julyatrika sari l200200101
#modul 10
#nomer 1b

import time
from timeit import timeit
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 3.8.6 (tags/v3.8.6:db45529, Sep 23 2020, 15:52:53) [MSC v
.1927 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more info
rmation.
>>>
========== RESTART: D:/MATKUL SMT 4/Praktikum ASD/modul 10/nomer
1b.py ==========
Jumlah adalah 500000500000, memerlukan 0.00000670 detik
Jumlah adalah 500000500000, memerlukan 0.00000420 detik
Jumlah adalah 500000500000, memerlukan 0.00000470 detik
Jumlah adalah 500000500000, memerlukan 0.00000730 detik
Jumlah adalah 500000500000, memerlukan 0.00000450 detik
>>>
```

   c. insertionSort

```
#Nicky Julyatrika sari l200200101
#modul 10
#nomer 1b

import random
from timeit import timeit
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
print("------average case scenario------")

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))

print("------worst case scenario------")
for i in range(5):
    siap = 'from __main__ import insertionSort, L'
    L = list(range(3000))
    L = L[::-1]
    t = timeit('insertionSort(L)', setup = siap, number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L), t))


print("------best case scenario------")
for i in range(5):
    siap = 'from __main__ import insertionSort, L'
    L = list(range(3000))
    t = timeit('insertionSort(L)', setup = siap, number=1)
    print("Mengurutkan %d bilangan, memerlukan %8.7f detik" % (len(L), t))
```

```
========== RESTART: D:/MATKUL SMT 4/Praktikum ASD/modul 10/nomer1c.py ==========
------average case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.9216733 detik
Mengurutkan 3000 bilangan, memerlukan 0.9467548 detik
Mengurutkan 3000 bilangan, memerlukan 0.9575016 detik
Mengurutkan 3000 bilangan, memerlukan 0.9374894 detik
Mengurutkan 3000 bilangan, memerlukan 1.0344786 detik
------worst case scenario------
Mengurutkan 3000 bilangan, memerlukan 1.8741298 detik
Mengurutkan 3000 bilangan, memerlukan 1.9424091 detik
Mengurutkan 3000 bilangan, memerlukan 1.7808753 detik
Mengurutkan 3000 bilangan, memerlukan 1.8064551 detik
Mengurutkan 3000 bilangan, memerlukan 1.8519997 detik
------best case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0008962 detik
Mengurutkan 3000 bilangan, memerlukan 0.0008001 detik
Mengurutkan 3000 bilangan, memerlukan 0.0018446 detik
Mengurutkan 3000 bilangan, memerlukan 0.0008026 detik
Mengurutkan 3000 bilangan, memerlukan 0.0008031 detik
>>>
```

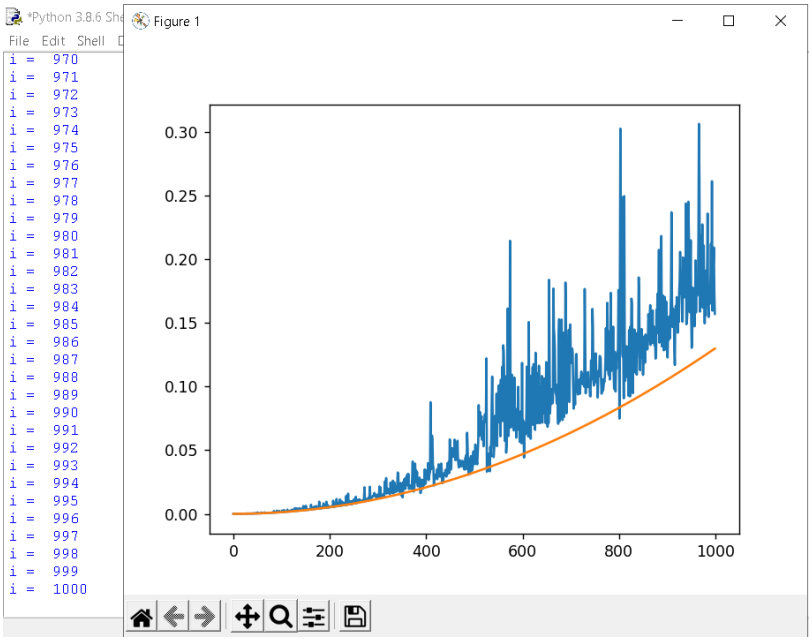
2. Python mempunyai perintah untuk mengurutkan suatu list yang memanfaatkan algoritma Timsort. Jika g adalah suatu list berisi bilangan, maka g.sort() kan mengurutkannya. Perintah yang lain, sorted() mengurutkan list dan mengembalikan sebuah list baru yang sudah urut. Selidikilah fungsi sorted ini menggunakan timeit:
    a. Apakah yang merupakan best case dan average case bagi sorted()?
    b. Confirm bahwa data input urutan terbalik bukan kasus terburuk bagi sorted(). Bahkan dia lebih cepat dalam mengurutkannya daripada data input random

```
#Nicky Julyatrika sari l200200101
#modul 10
#nomer 2

from timeit import timeit
import random

print("------average case scenario------")
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("------worst case scenario------")
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))

print("------best case scenario------")
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))
```

```
========== RESTART: D:/MATKUL SMT 4/Praktikum ASD/modul 10/nomer2.py ==========
------average case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0015543 detik
Mengurutkan 3000 bilangan, memerlukan 0.0013639 detik
Mengurutkan 3000 bilangan, memerlukan 0.0013556 detik
Mengurutkan 3000 bilangan, memerlukan 0.0013252 detik
Mengurutkan 3000 bilangan, memerlukan 0.0020544 detik
------worst case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0000994 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001010 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001099 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001039 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001002 detik
------best case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0001418 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000913 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001155 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000952 detik
Mengurutkan 3000 bilangan, memerlukan 0.0001140 detik
>>>
```

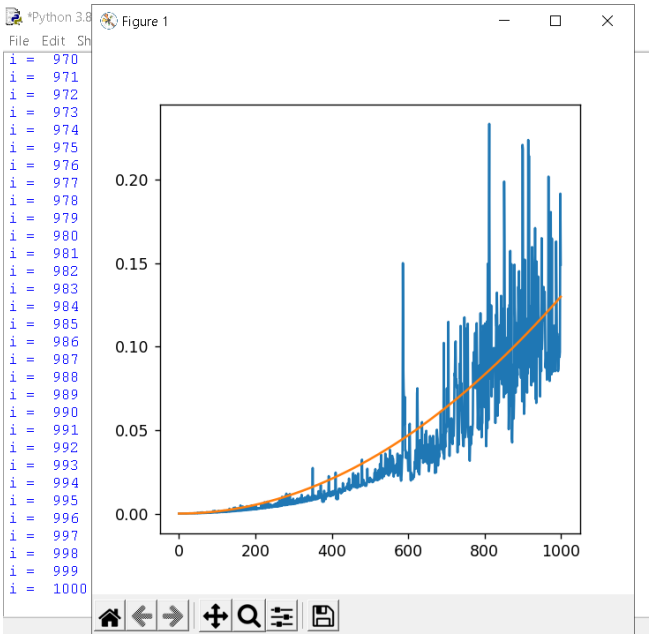
3. Untuk tiap kode berikut, tentukan running time-nya, O(1), O(log n), O(n), O(n log n), O(n2), atau O(n3), atau yang lain. Untuk memulai analisis, ambil suatu nilai n tertentu, lalu ikuti apa yang terjadi di kode itu.
    a. loop di dalam loop, keduanya sebanyak n:

```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 3a

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    test = 0
    for i in range(n):
        for j in range(n):
            test = test + i * j
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 1000
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



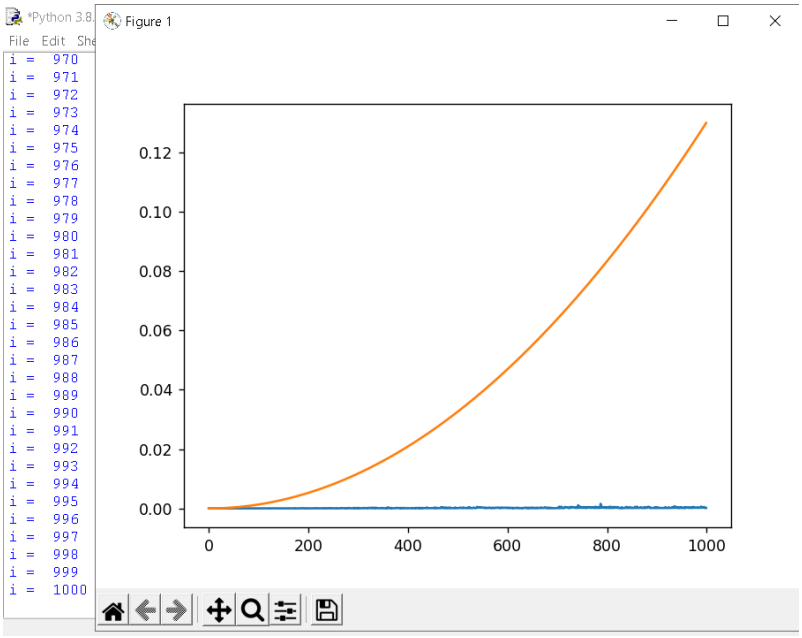
b. loop di dalam loop, yang dalam bergantung nilai i loop luar:

```
#Nicky Julyatrika sari 1200200101
#modul 10
#nomer 3b

import timeit
import matplotlib.pyplot as plt

def kalangBersusuh(n):
    test = 0
    for i in range(n):
        for j in range(i):
            test = test + i * j
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 1000
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



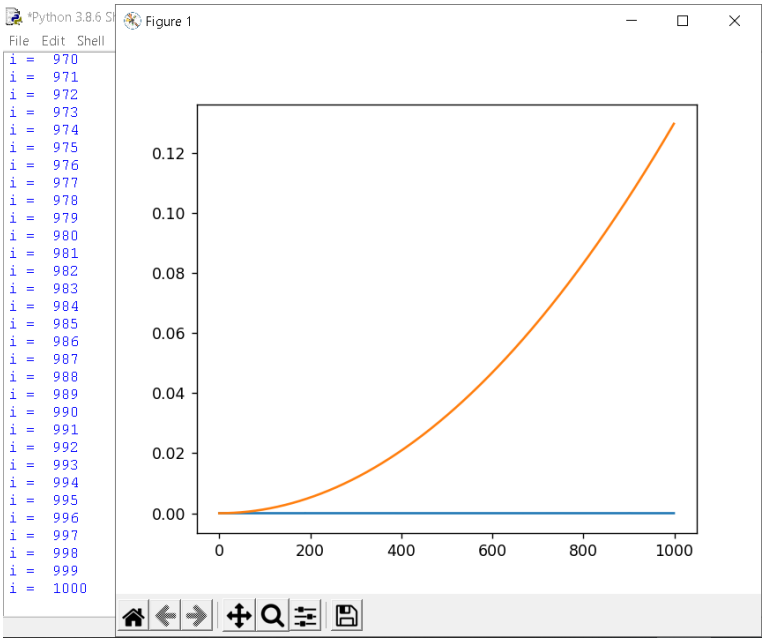
c. dua loop terpisah:

```
#Nicky Julyatrika sari 1200200101
#modul 10
#nomer 3c

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    test = 0
    for i in range(n):
        test = test + 1
    for j in range(n):
        test = test - 1

def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 1000
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



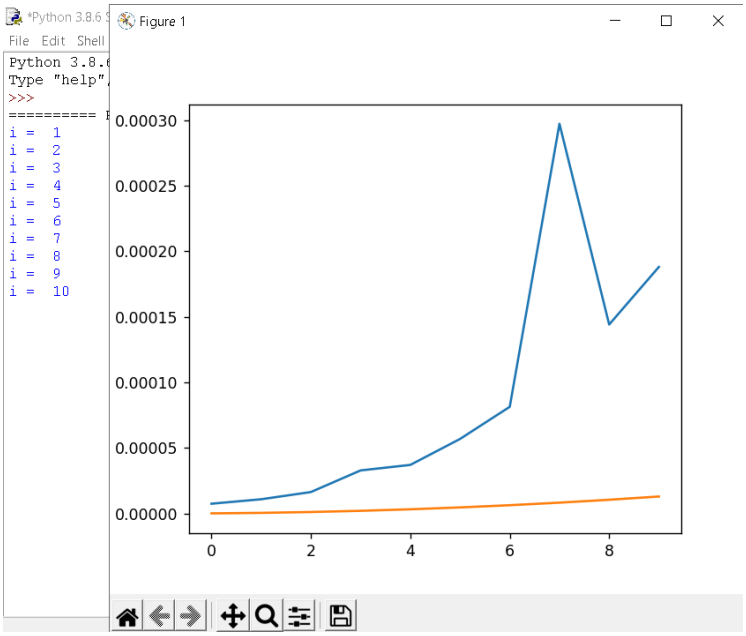
d. while loop yang dipangkas separuh tiap putaran

```
#Nicky Julyatrika sari 1200200101
#modul 10
#nomer 3d

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    i = n
    while i > 0:
        k = 2 + 2
        i = i // 2
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 1000
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



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

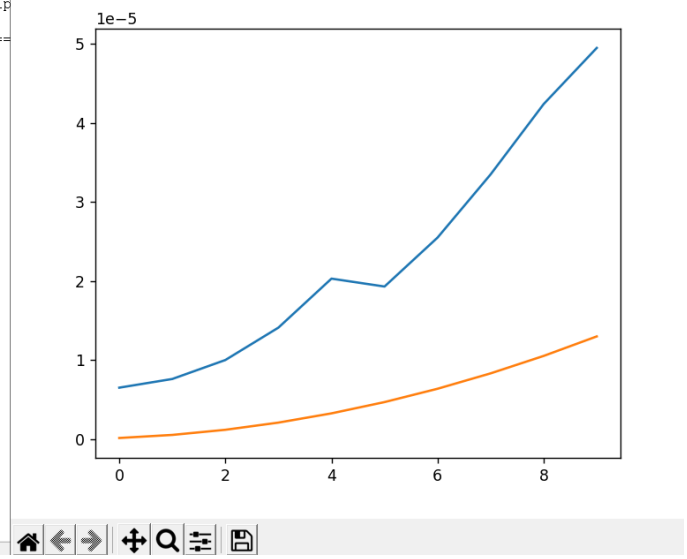
```
#Nicky Julyatrika sari 1200200101
#modul 10
#nomer 3e

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    for i in range(n):
        for j in range(n):
            for k in range(n):
                m = i + j + k + 2019
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 10
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



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

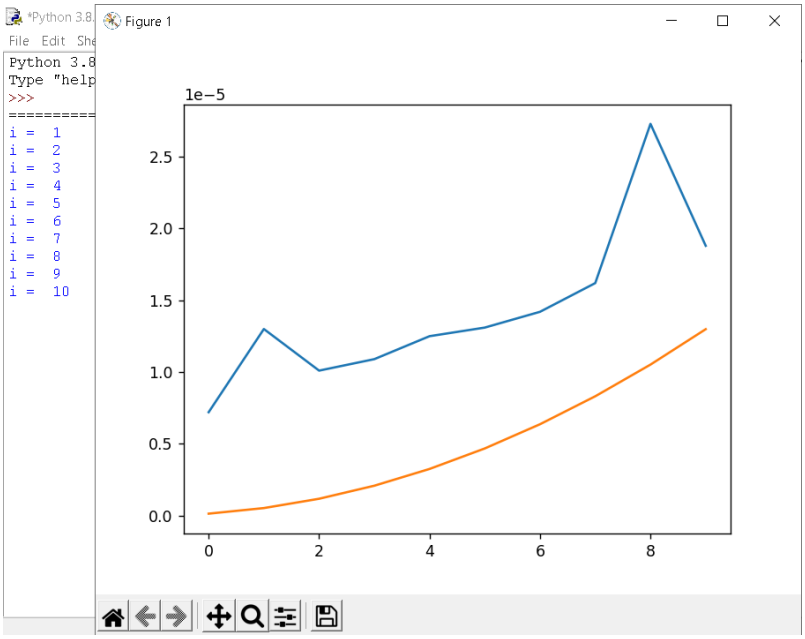
```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 3f

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    for i in range(n):
        for j in range(i):
            for k in range(j):
                m = i + j + k + 2019
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 10
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```

```
Python 3.8
Type "help
>>>
==========
i =  1
i =  2
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
i =  9
i =  10
```



g. fungsi ini:

```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 3g

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(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 ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 10
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



4. Urutkan dari yang pertumbuhan kompleksitasnya lambat ke yang cepat:
   $log_4 n < 10log_2 n < n\,log_2 n < 2\,log_2 n < 5n^2 < n^3 < 12n^6 < 4^n$

5. Tentukan O(·) dari fungsi-fungsi berikut, yang mewakili banyaknya langkah yang diperlukan untuk beberapa algoritm
   a. $T(n) = n^2 + 32n + 8 = O(n^2)$
   b. $T(n) = 87n + 3n = O(n)$
   c. $T(n) = 4n + 5n\log n + 102 = O(n\ \log n)$
   d. $T(n) = \log n + 3n^2 + 88 = O(n^2)$
   e. $T(n) = 3(2n) + n^2 + 647 = O(2^n)$
   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. (Literature Review) Carilah di Internet, kompleksitas metode-metode pada object list di Python. Hint:
   a. Google python list methods complexity . Lihat juga bagian ”Images”-nya
   b. Kunjungi https://wiki.python.org/moin/TimeComplexit

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

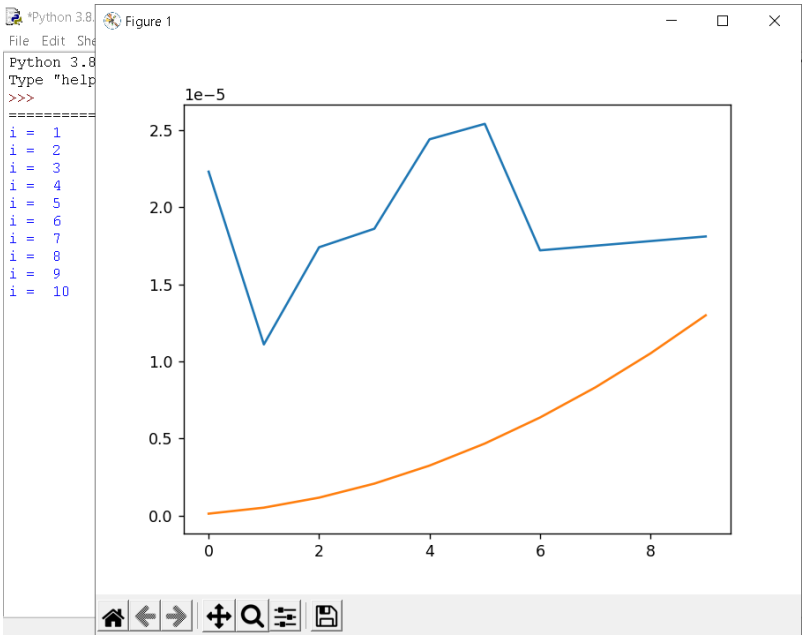
   Secara internal, list direpresentasikan sebagai array; biaya terbesar berasal dari pertumbuhan di luar ukuran alokasi saat ini (karena semuanya harus bergerak), atau dari memasukkan atau menghapus suatu tempat di dekat awal (karena semuanya setelah itu harus bergerak). Jika perlu kita menambahkan/menghapus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

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(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)

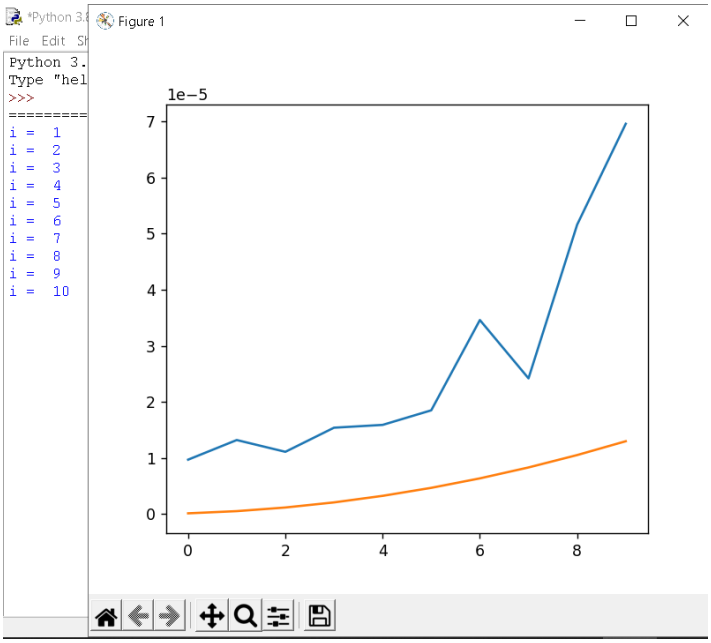
7. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode append() adalah O(1). Gunakan timeit dan matplotlib, seperti sebelumnya.

```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 7

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    L = list(range(30))
    L = L[::-1]
    for i in range(n):
        L.append(n)
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 10
LS = ujiKalangBersusuh(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 matplotlib, seperti sebelumnya.

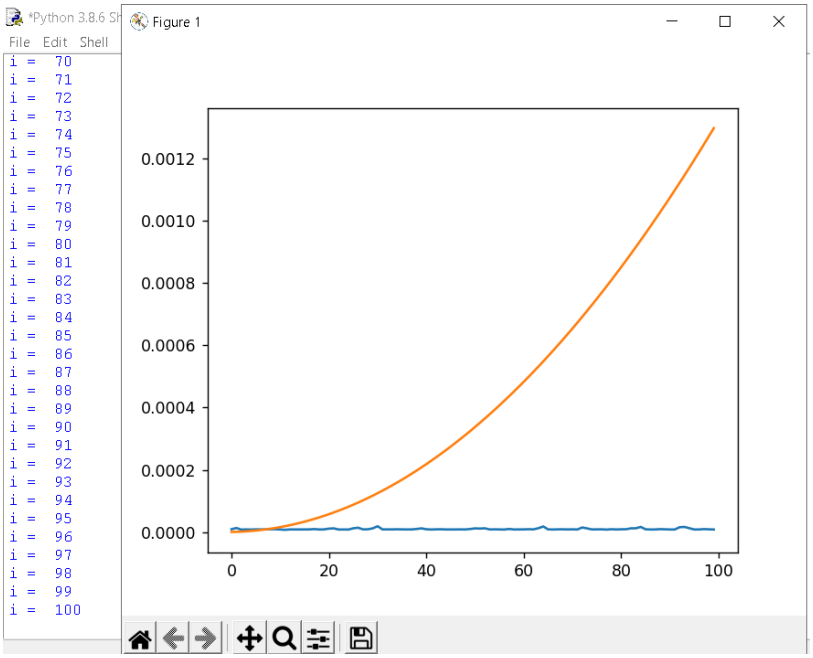
```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 8

import timeit
import matplotlib.pyplot as plt
def kalangBersusuh(n):
    L = list(range(30))
    L = L[::-1]
    for i in range(n):
        for b in range(n):
            L.insert(i,b)
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 10
LS = ujiKalangBersusuh(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-nilaiberada-di-suatu-list mempunyai kompleksitas O(n). Gunakan timeit dan matplotlib, seperti sebelumnya.

```
#Nicky Julyatrika sari l20020010
#modul 10
#nomer 9

import timeit
import time
import matplotlib.pyplot as plt
def straight(cont, target):
    n = len(cont)
    for i in range(n):
        if cont[i] == target:
            return True
    return False
def tim():
    a = 100
    b = [12, 3, 5, 6, 8, 2, 11]
    awal = time.time()
    U = straight(b, a)
    akhir = time.time()
    print('Worst case')
    print('mengurutkan %d bilangan, memerlukan %8.7f detik' % (U, akhir-awal))
```

```
tim()
def kalangBersusuh(n):
    a = 100
    b = [12, 3, 5, 6, 8, 2, 11]
    U = straight(b, n)
def ujiKalangBersusuh(n):
    ls = []
    jangkauan = range(1, n+1)
    siap = 'from __main__ import kalangBersusuh'
    for i in jangkauan:
        print('i = ', i)
        t = timeit.timeit('kalangBersusuh(' + str(i) +')', setup = siap, number = 1)
        ls.append(t)
    return ls
n = 100
LS = ujiKalangBersusuh(n)
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1, n+1)])
plt.show()
```



10. (Literature Review) 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.

    Perhatikan bahwa ada jalur cepat untuk dict yang (dalam praktiknya) hanya menangani kunci str; ini tidak mempengaruhi kompleksitas algoritmik, tetapi secara signifikan dapat mempengaruhi faktor konstan: seberapa cepat program biasa selesai.

Operation	Average Case	Amortized Worst Case
k in d	O(1)	O(n)
Copy[3]	O(n)	O(n)
Get Item	O(1)	O(n)
Set Item[1]	O(1)	O(n)
Delete item	O(1)	O(n)
Iteration[3]	O(n)	O(n)

.

11. (Literature Review) Selain notasi big-O O(·), ada pula notasi big-Theta Θ(·) dan notasi big-Omega Ω(·). Apakah beda di antara ketiganya?
    a. 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.

b. 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.
c. 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. (Literature Review) Apa yang dimaksud dengan amortized analysis dalam analisis algoritma?

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 dilakuka

Dalam Amortized Analysis,waktu yang diperlukan untuk memproses struktur-data terurut dirata-ratakan dari semua operasi.Amortized Analysis dapat digunakan untuk menunjukkan bahwa nilai operasi rata-rata cukup kecil, walaupun ada beberapa operasi yang memiliki nilai besar. Metode ini cukup berguna untuk menghitung nilai sumber daya yang terbuang secara keseluruhan.