LAPORAN

PRAKTIKUM ALGORITMA DAN STRUKTUR DATA

(MODUL 10)

“ Analisis Algoritma”

Disusun oleh :

NAMA : CINDI DILA APRILIANA

NIM : L200200106

KELAS : E

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

Nomor 1.py - C:\Users\MSI GF63\Documents\UMS\TUGAS SEMESTER 4\Praktikum Algoritma...

File Edit Format Run Options Window Help

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106


#NOMOR 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))
```

Output

```
>>>
= RESTART: C:\Users\MSI GF63\Documents\UMS\TUGAS SEMESTER 4\I
MODUL 10\Nomor 1.py
Jumlah adalah 500000500000, memerlukan 0.03749790 detik
Jumlah adalah 500000500000, memerlukan 0.04439960 detik
Jumlah adalah 500000500000, memerlukan 0.03556700 detik
Jumlah adalah 500000500000, memerlukan 0.04657480 detik
Jumlah adalah 500000500000, memerlukan 0.03635710 detik
>>>
```

b) jumlahkan_cara_2

```
#NOMOR 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))
```

Output

```
>>>
= RESTART: C:\Users\MSI GF63\Documents\UMS\TUGAS SEMESTER 4\
MODUL 10\Nomor 1.py
Jumlah adalah 500000500000, memerlukan 0.00000480 detik
Jumlah adalah 500000500000, memerlukan 0.00000460 detik
Jumlah adalah 500000500000, memerlukan 0.00000450 detik
Jumlah adalah 500000500000, memerlukan 0.00000430 detik
Jumlah adalah 500000500000, memerlukan 0.00000420 detik
>>>
```

c) insertionSort

```
#NOMOR 1C
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))
```

Output

```
MODUL 10\Nomor 1.py
------average case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.3565097 detik
Mengurutkan 3000 bilangan, memerlukan 0.3252978 detik
Mengurutkan 3000 bilangan, memerlukan 0.4055579 detik
Mengurutkan 3000 bilangan, memerlukan 0.3467713 detik
Mengurutkan 3000 bilangan, memerlukan 0.5215998 detik
------worst case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.7025959 detik
Mengurutkan 3000 bilangan, memerlukan 0.8199940 detik
Mengurutkan 3000 bilangan, memerlukan 0.9321969 detik
Mengurutkan 3000 bilangan, memerlukan 0.8278643 detik
Mengurutkan 3000 bilangan, memerlukan 1.0892976 detik
------best case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0004145 detik
Mengurutkan 3000 bilangan, memerlukan 0.0002711 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003227 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003214 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003761 detik
>>>
```

2. Algoritma timsort

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106

#nomor 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))
```

Output

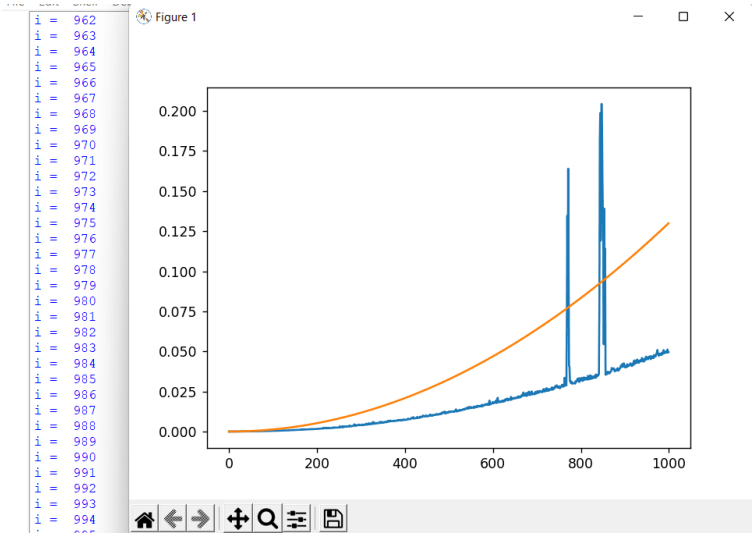
```
MODUL 10/Nomor 2.py
------average case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0004978 detik
Mengurutkan 3000 bilangan, memerlukan 0.0005721 detik
Mengurutkan 3000 bilangan, memerlukan 0.0002698 detik
Mengurutkan 3000 bilangan, memerlukan 0.0002805 detik
Mengurutkan 3000 bilangan, memerlukan 0.0003125 detik
------worst case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0000360 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000264 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000411 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000296 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000237 detik
------best case scenario------
Mengurutkan 3000 bilangan, memerlukan 0.0000209 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000210 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000205 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000242 detik
Mengurutkan 3000 bilangan, memerlukan 0.0000479 detik
>>>
```

3. Tentukan running time-nya, O(1), O(log n), O(n), O(n log n), O(n2), atau O(n3)

a) Loop di dalam loop, keduanya sebanyak

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106

#NOMOR 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()
```

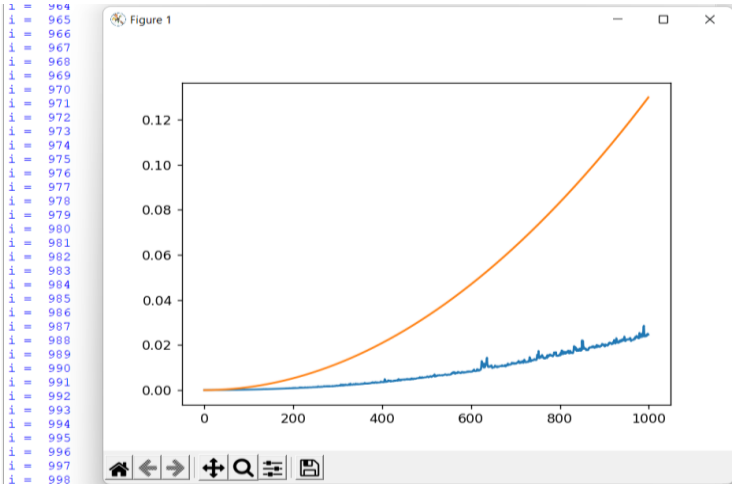
Output



b) Loop di dalam loop, yang dalam bergantung n

```
#NOMOR 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()
```

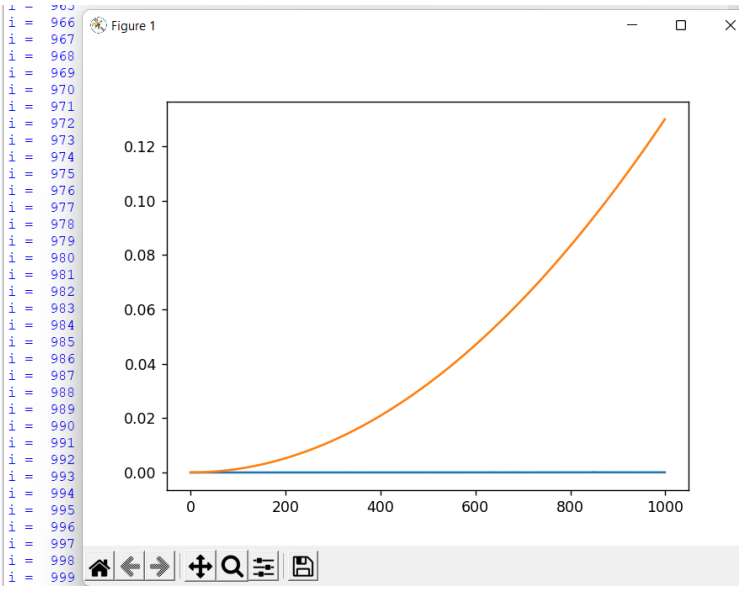
Output



c) Dua loo

```
#NOMOR 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()
```

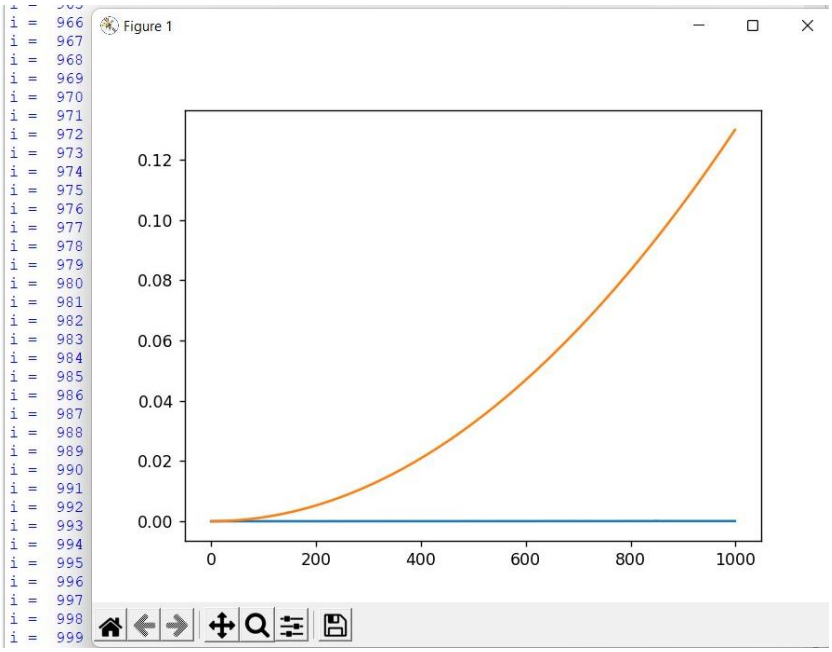
Output



d) While loop yang dipangkas separuh tiap putaran

```
#NOMOR 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()
```

Output

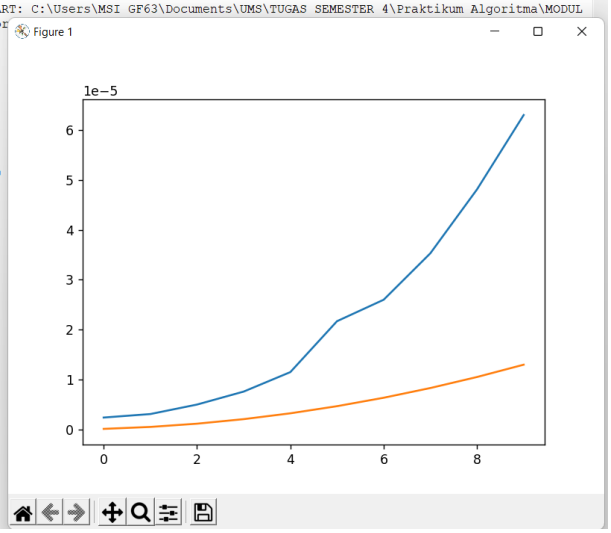


e) Loop di dalam loop, ketiganya sebanyak n

```
#NOMOR 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()
```

Output

```
>>>
= RESTART: C:\Users\MSI GF63\Documents\UMS\TUGAS SEMESTER 4\Praktikum Algoritma\MODUL
10\Nomor
i =  1
i =  2
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
i =  9
i =  10
```

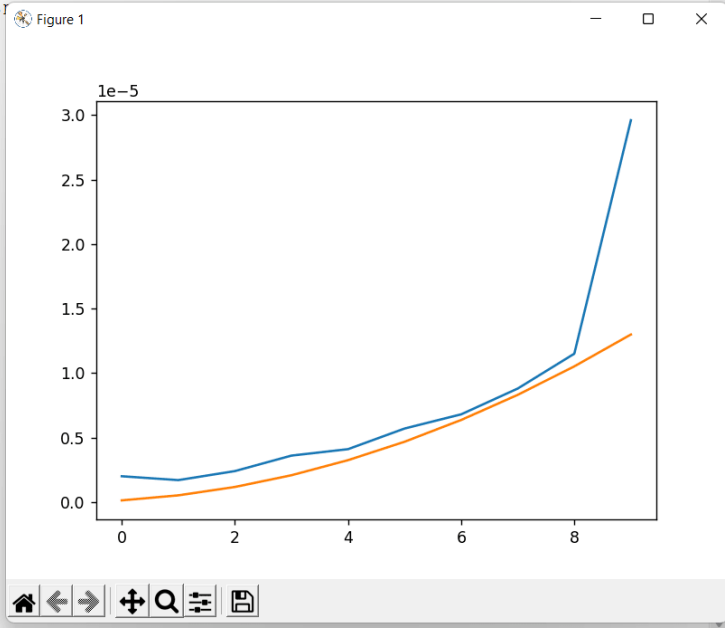


f) Loop di dalam loop, dengan loop dalam sebanyak nilai loop luar terdekat

```
#NOMOR 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()
```

Output

```
= RESTART: C:\Users\MSI GF63\Documents\UMS\TUGAS SEMESTER 4\Praktikum Algoritma\MODUL 1
0\Nomor 3.
i =  1
i =  2
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
i =  9
i =  10
```

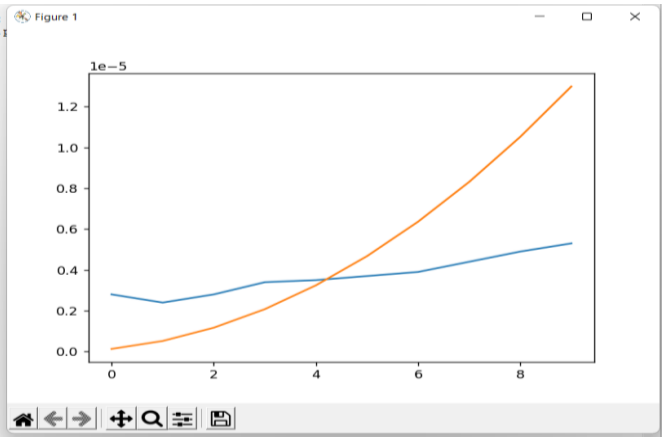


g) Fungsi i

```
#NOMOR 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()
```

Output

```
>>>
= RESTART:
0\Nomor 3.p
i =  1
i =  2
i =  3
i =  4
i =  5
i =  6
i =  7
i =  8
i =  9
i =  10
```



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(.) dari fungsi-fungsi berikut, yang mewakili banyaknya Langkah yang diperlukan 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. $T(n) = 3(2n) + n^2 + 647 = O(2^n)$
   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. Carilah di internet, kompleksitas metode-metode pada object list di Python. Hint
   - Google ‘Python list methods complexity’. Lihat juga bagian ‘images’
   - Kunjungi https://wiki.python.org/moin/TimeComplexity

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

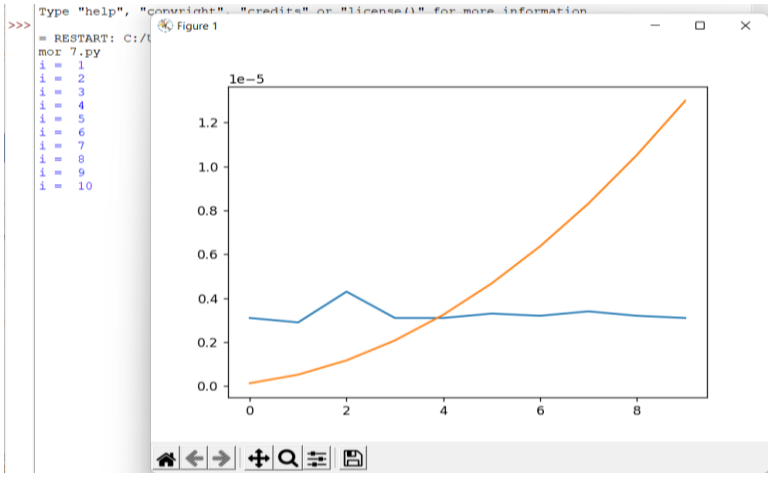
Secara internal, list direpresentasikan sebagai array; biaya terbesar berasal dari pertumbuhan di luar ukuran alokasi saat ini (karena semuanya harus bergerak), atau dari

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.

7. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode append() adalah O(1).

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106

#NOMOR 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()
```

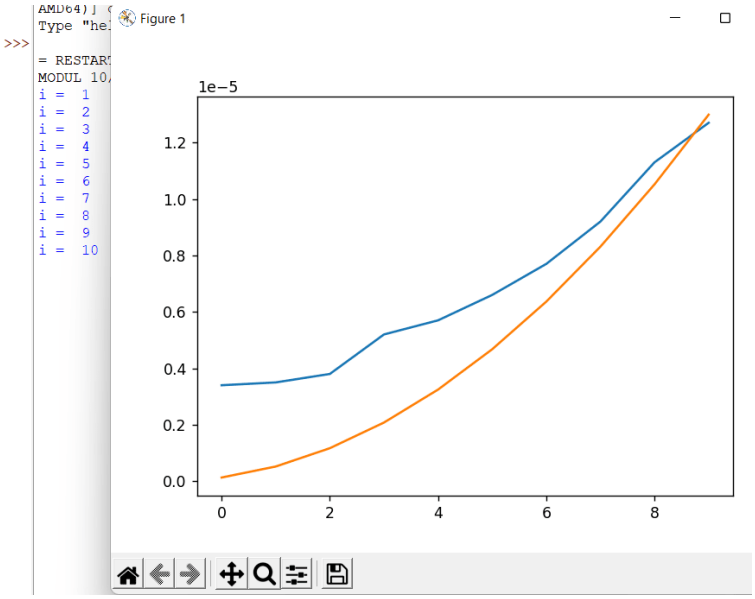
Output



8. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa metode insert() adalah O(n).

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106

#NOMOR 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()
```

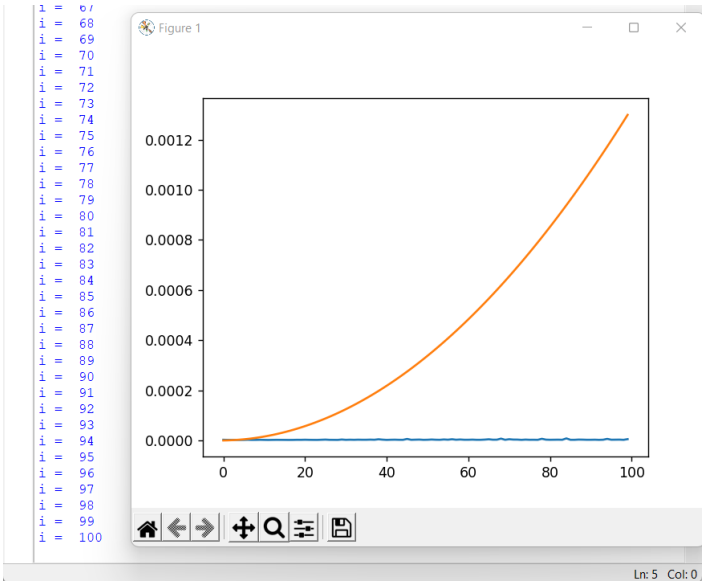
Output



9. Buatlah suatu ujicoba untuk mengkonfirmasi bahwa untuk memeriksa apakah-suatu-nilai-berada-di-suatu-list mempunyai kompleksitas O(n).

```
#MODUL 10
#CINDI DILA APRILIANA_L200200106

#NOMOR 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
        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()
```

Output



10. 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.

11. Perbedaan notasi big-O , big-Theta dan notasi big-Omega:
    - 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. 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