**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))

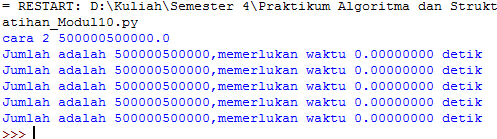
for i in range(5):

awal = time.time()

h = jumlahkan\_cara\_2(1000000)

akhir = time.time()

print("Jumlah adalah %d,memerlukan waktu %9.8f detik" % (h,akhir-awal))



###### 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):

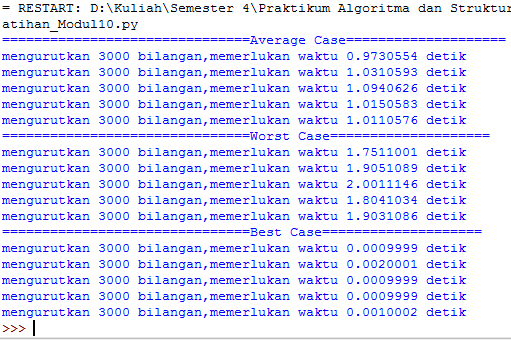
L = list(range(3000))

awal = time.time()

U = insertionsort(L)

akhir = time.time()

print("mengurutkan %d bilangan,memerlukan waktu %8.7f detik" % (len(L),akhir-awal))



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

###########################################

x = 5

y = x

z = x + y\*8

d = x > 0 and x < 100

f = [3,2,4,5]

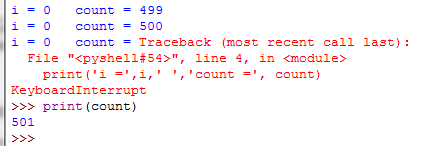
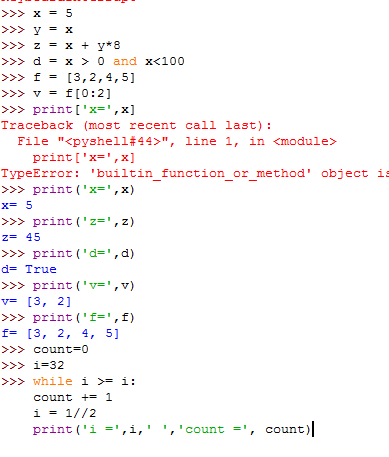
v = f[0:2]

i = 32

while i >= i:

count += 1

i = 1//2

print('i =',i,' ','count =', count)  


##### 10.5 Analisis pewaktuan menggunakan timeit #####

######################################################

from timeit import timeit

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

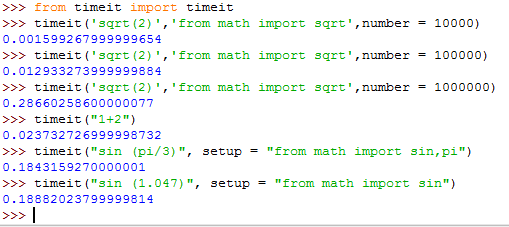
timeit('sqrt(2)','from math import sqrt',number = 100000)

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")



#######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.

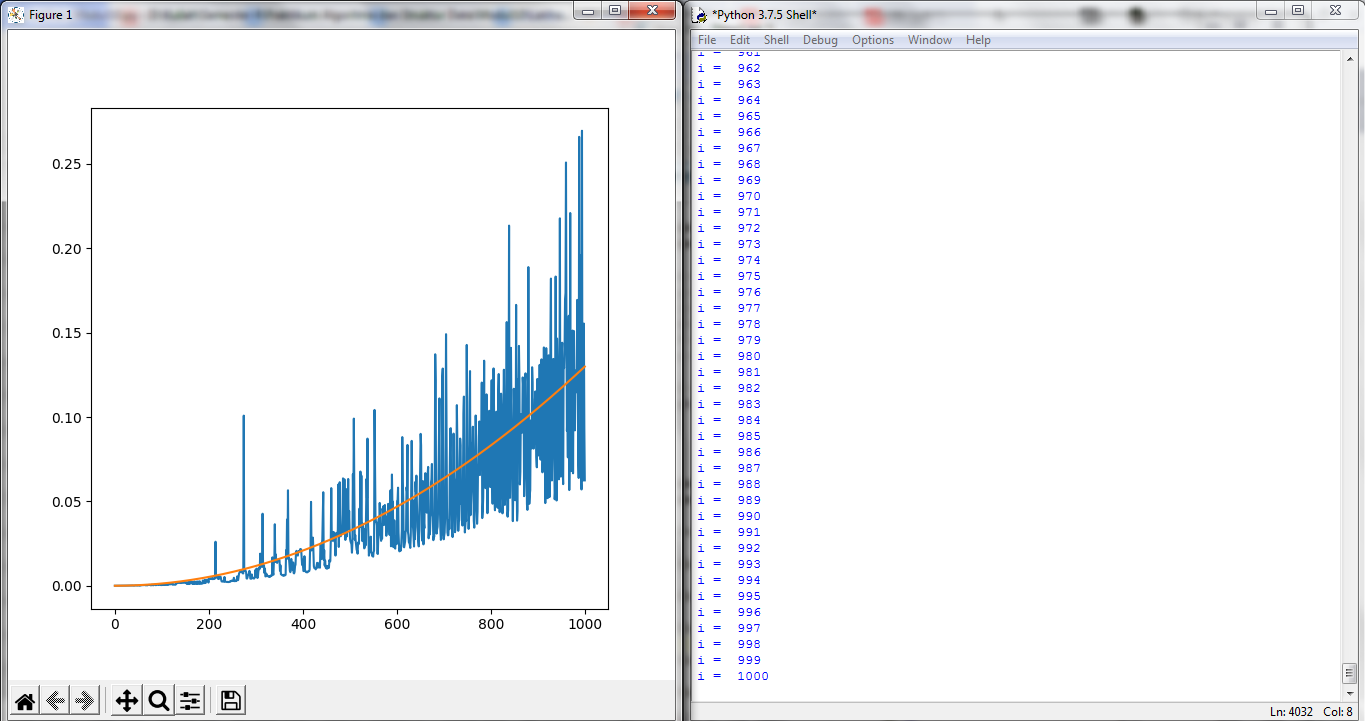
## Menggambar grafik. dibawah ini saja yang diulang saat me-nyetel skala.

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

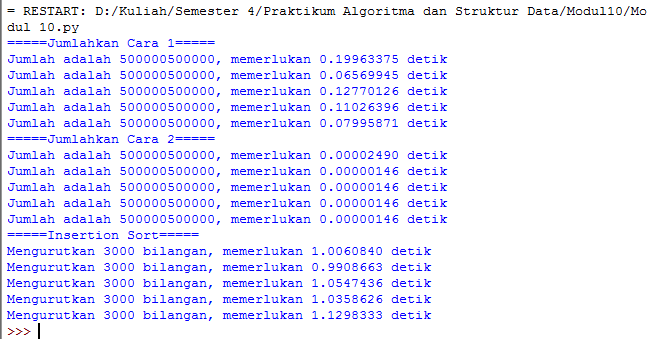
for i in range(5):

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))  
  
  
**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))

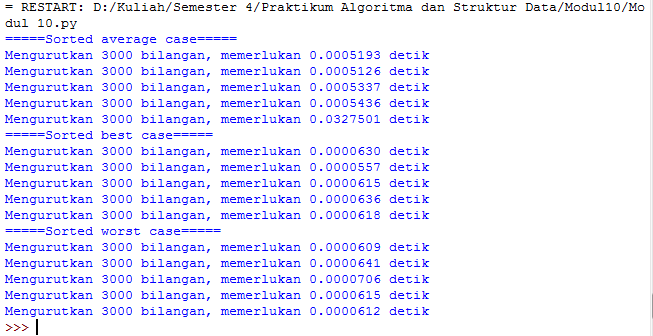
print("=====Sorted worst case=====")

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))  
  
  
  
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

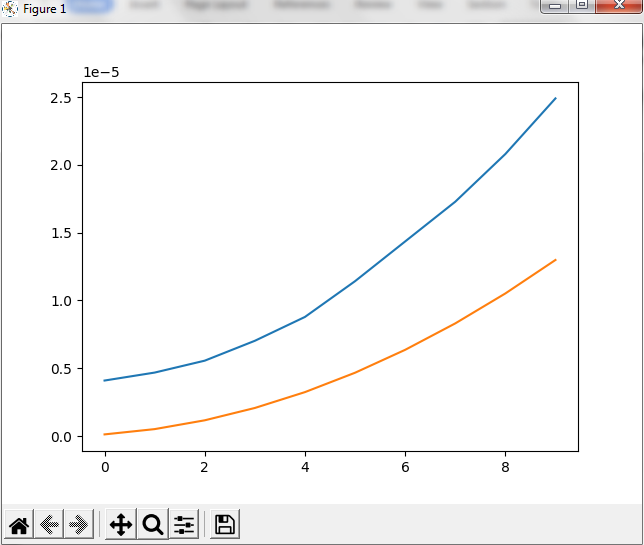
n = 10

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

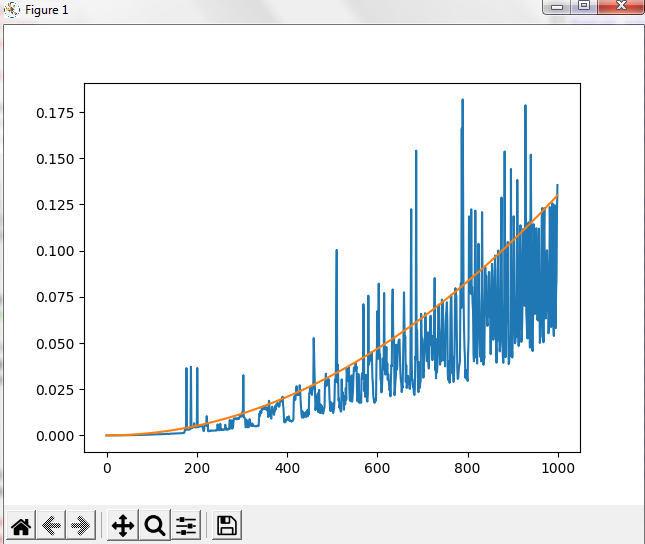
n = 1000

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

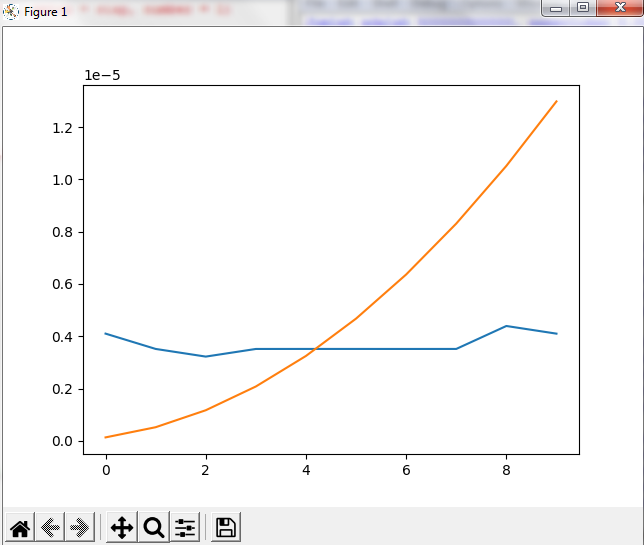
n = 10

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

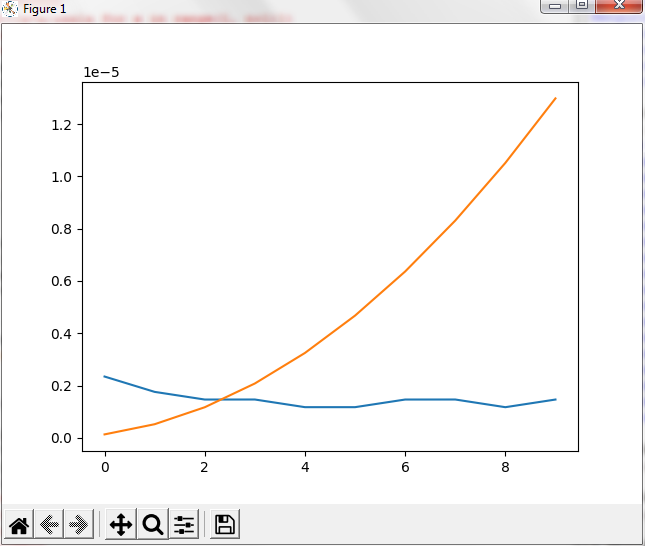
n = 10

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

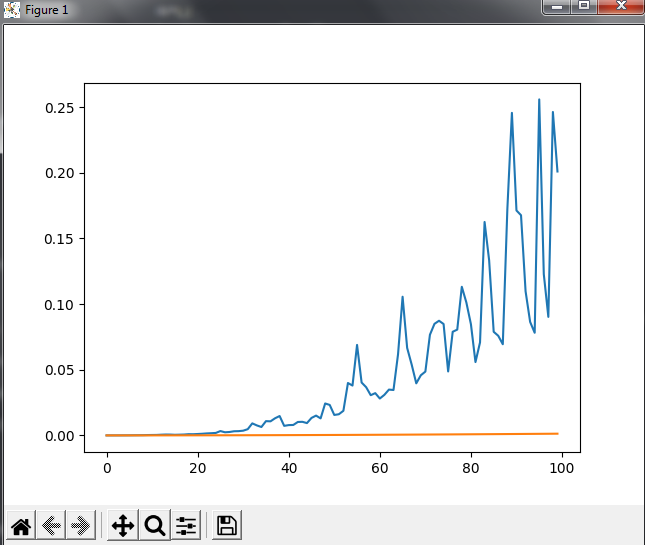
n = 100

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

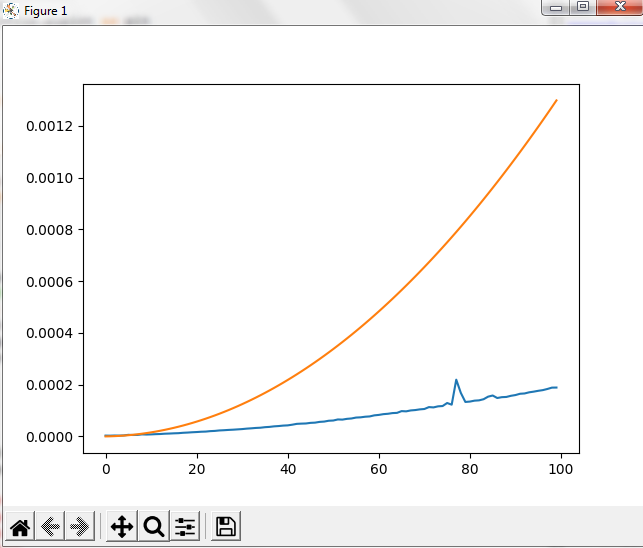
n = 100

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

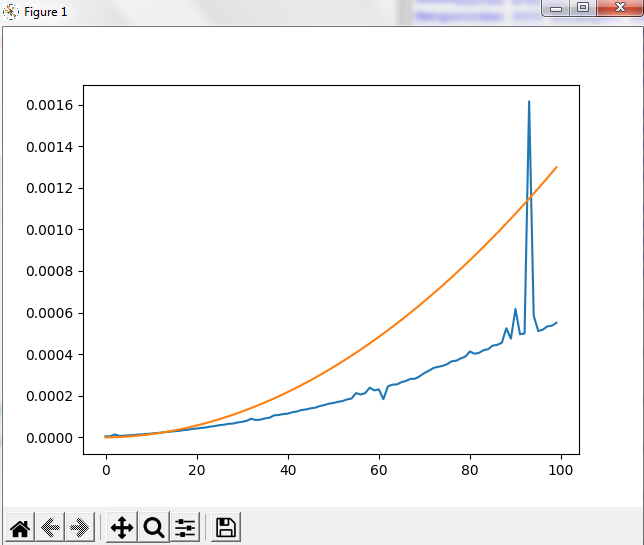
n = 100

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)

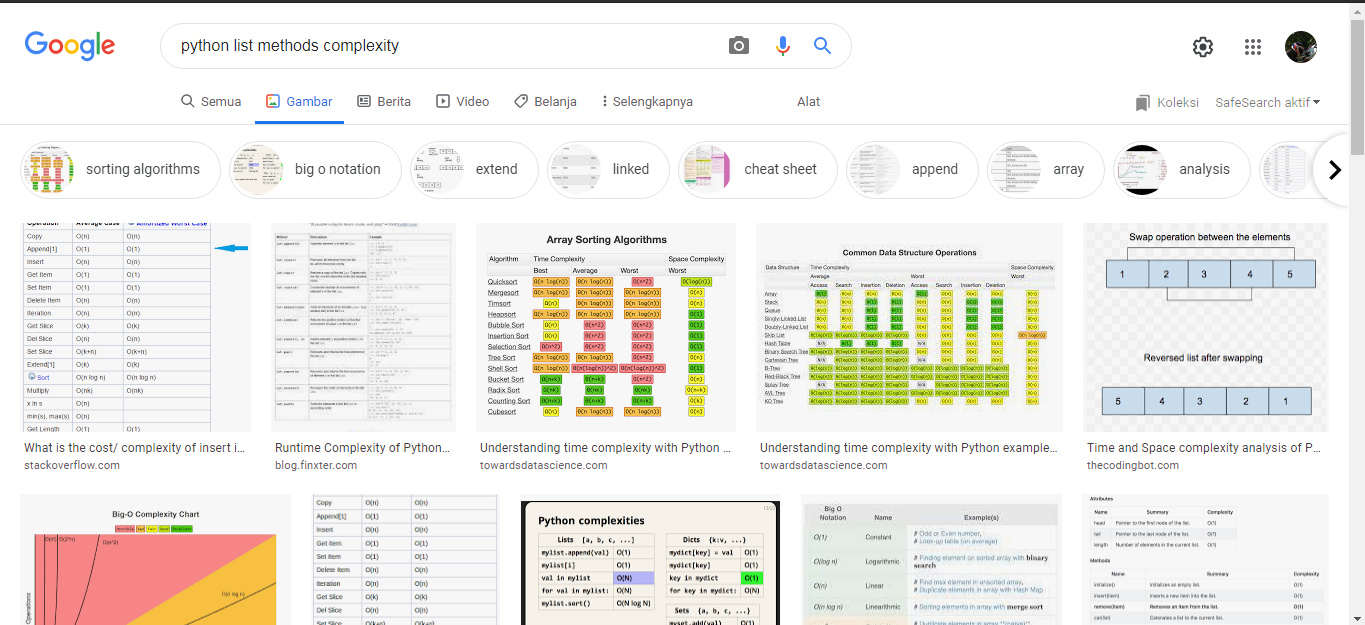
(e) T(n) = 3(2^n) + n^2 + 647 = O(2^n)

(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)

1. **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>  
  
  
**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

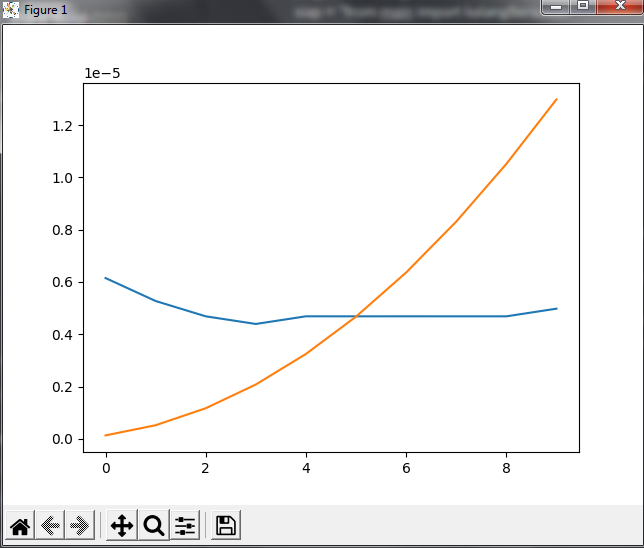
n = 10

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

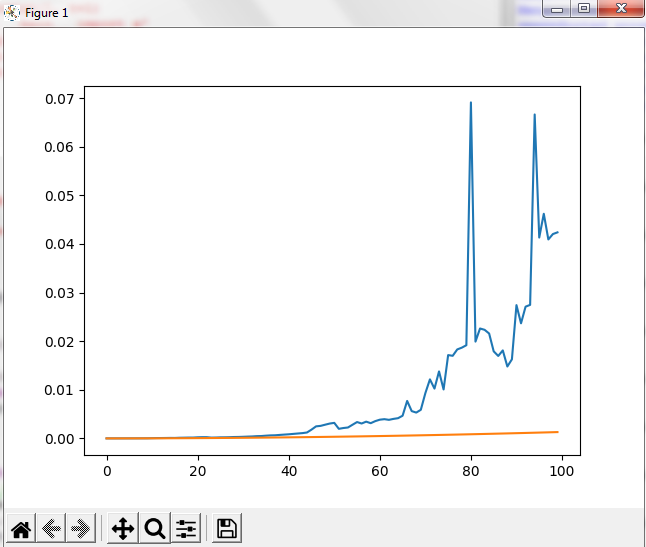
n = 100

LS = ujia(n)

plt.plot(LS)

skala = 7700000

plt.plot([x\*x/skala for x in range(1, n+1)])

plt.show()

1. **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

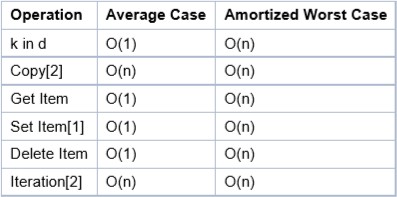
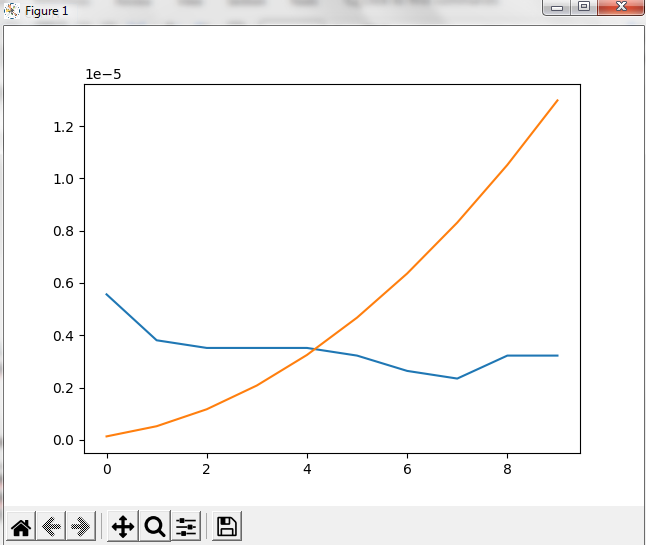
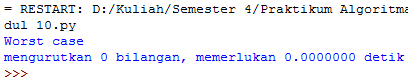
n = 10

LS = ujia(n)

plt.plot(LS)

skala = 7700000

plt.plot([x\*x/skala for x in range(1, n+1)])

plt.show()  
|  
  
**10. Carilah di internet, kompelsitas metode pada object dict di python.**  
  
**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