

LAPORAN PRAKTIKUM
ALGORITMA DAN STRUKTUR DATA
LANJUTAN MODUL 10
ANALISIS ALGORITMA



Disusun oleh:
Bima Triadmaja
L200210137
E

TEKNIK INFORMATIKA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA
2022/2023

Soal-soal untuk mahasiswa

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

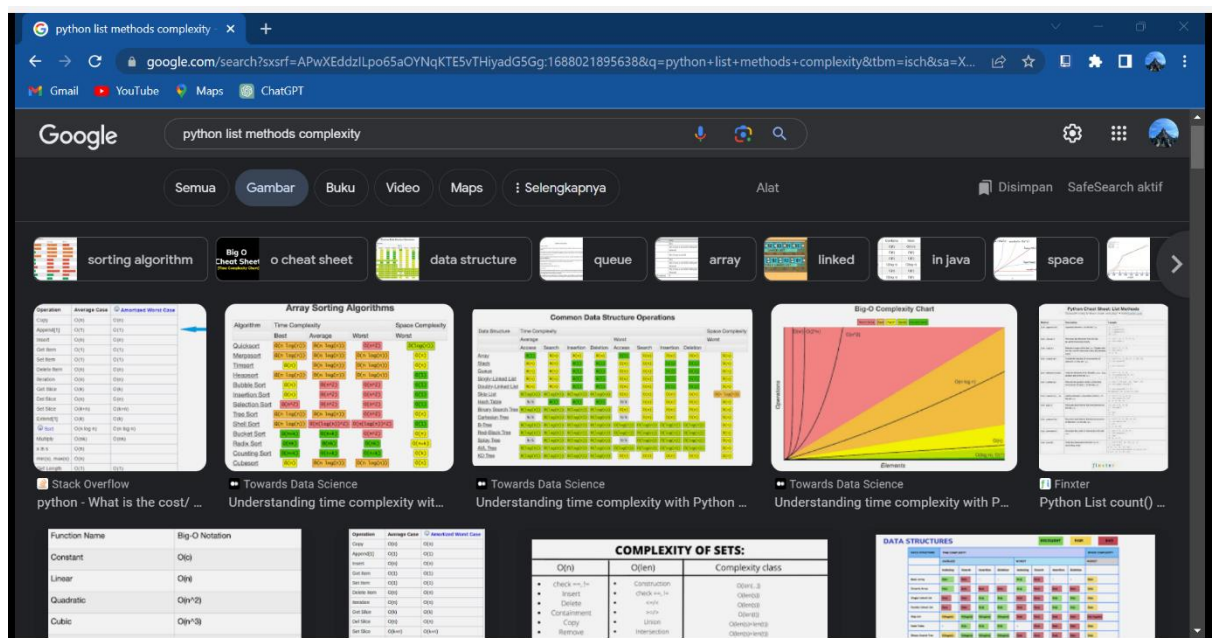
$\log 4n$; $10 \log 2n$; $n \log 2n$; $2 \log 2n$; $5n^2$; n^3 ; $12n^6$; $4n$

5. Tentukan $O(\cdot)$ dari fungsi-fungsi berikut yang mewakili banyaknya langkah yang diperlukan untuk beberapa algoritma.

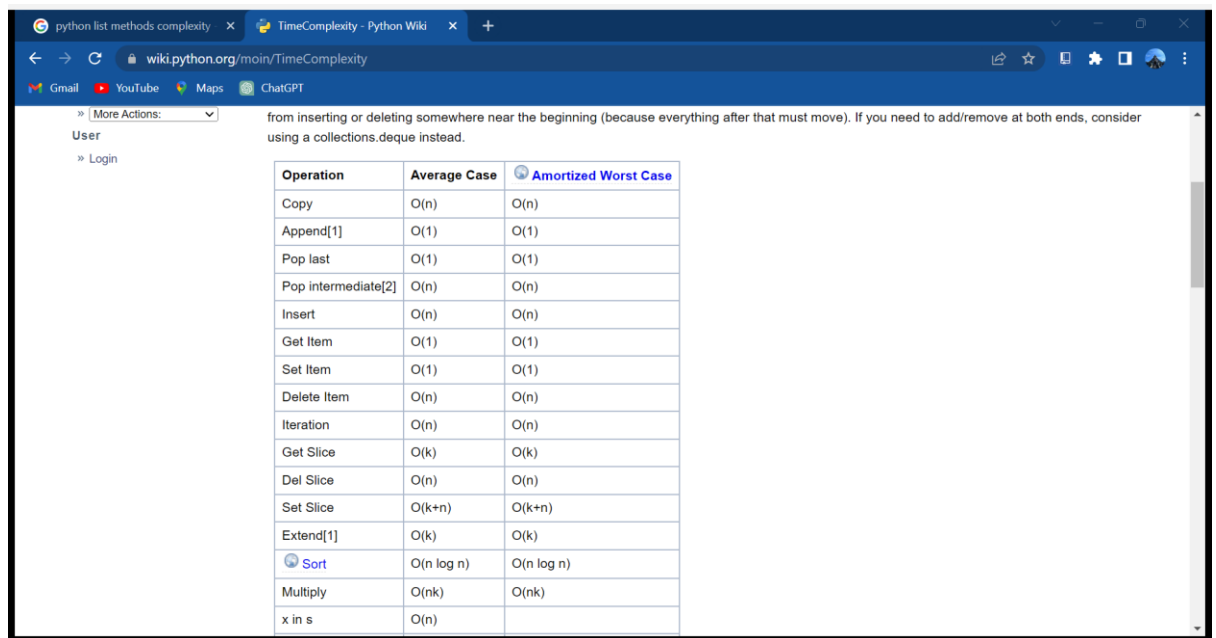
- $T(n) = n^2 + 32n + 8 = O(n^2)$
- $T(n) = 87n + 8n = O(n)$
- $T(n) = 4n + 5n \log n + 102 = O(n \log n)$
- $T(n) = \log n + 3n^2 + 88 = O(n^2)$
- $T(n) = 3(2^n) + n^2 + 647 = O(2^n)$
- $T(n, k) = kn + \log k = O(kn)$
- $T(n, k) = 8n + k \log n + 800 = O(n)$
- $T(n, k) = 100kn + n = O(kn)$

6. Carilah di internet, kompleksitas metode-metode pada object list di Python.


- a. Google **python list methods complexity** . Lihat juga bagian "Images"-nya



b. Kunjungi <https://wiki.python.org/moin/TimeComplexity>



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[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	$O(n)$	

7. Ujicoba untuk mengkonfirmasi bahwa metode `append()` adalah $O(1)$.

```
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)
```

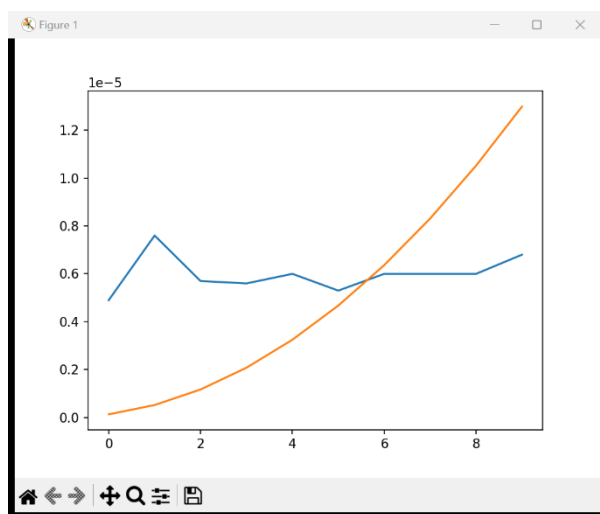
```

        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()
print('')
print('Sudah selesai.')

print('\n--- Oleh L200210137 ---')
```



8. Ujicoba untuk mengkonfirmasi bahwa metode insert() adalah $O(n)$.

```

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):
```

```

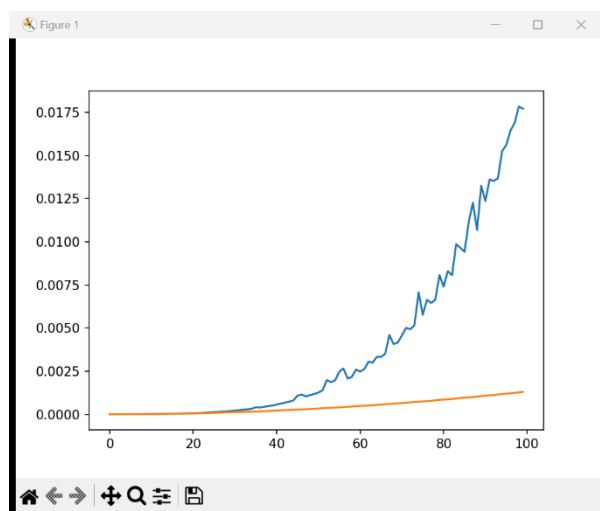
        for b in range(n):
            L.insert(i,b)

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()
print('')
print('Sudah selesai.')

print('\n--- Oleh L200210137 ---')
```



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

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

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

def timee():
    n = 200
    lis = [1,6,7,8,3,2,11]
    awal = time.time()
    U = cari(lis, n)
    akhir = time.time()
    print("Jumlah adalah %d, memerlukan %8.7f detik" % (U,
akhir-awal))
timee()

def a(n):
    a = [1,6,7,8,3,2,11]
    U = cari(a,n)

def ujia(n):
    ls=[]
    jangkauan = range(1,n+1)
    for i in jangkauan:
```

```

        t = timeit.timeit("a(" + str(i) + ")", "from __main__
import a", number=1)
        ls.append(t)
    return ls

```

```

n = 10
LS = ujia(n)

```

```

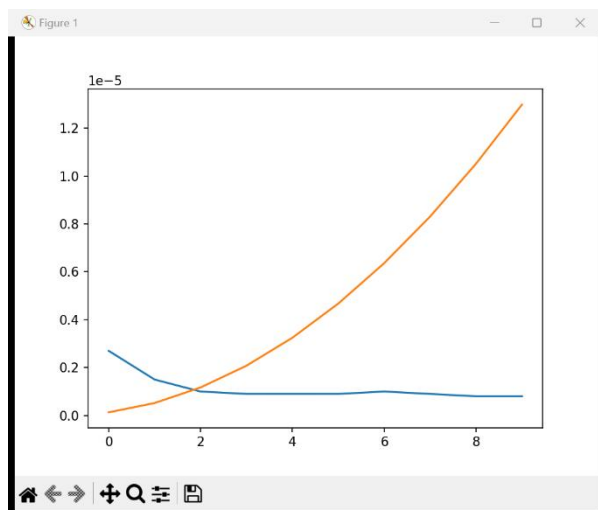
plt.plot(LS)
skala = 7700000
plt.plot([x*x/skala for x in range(1,n+1)])
plt.show()
print('')
print('Sudah selesai.')

```

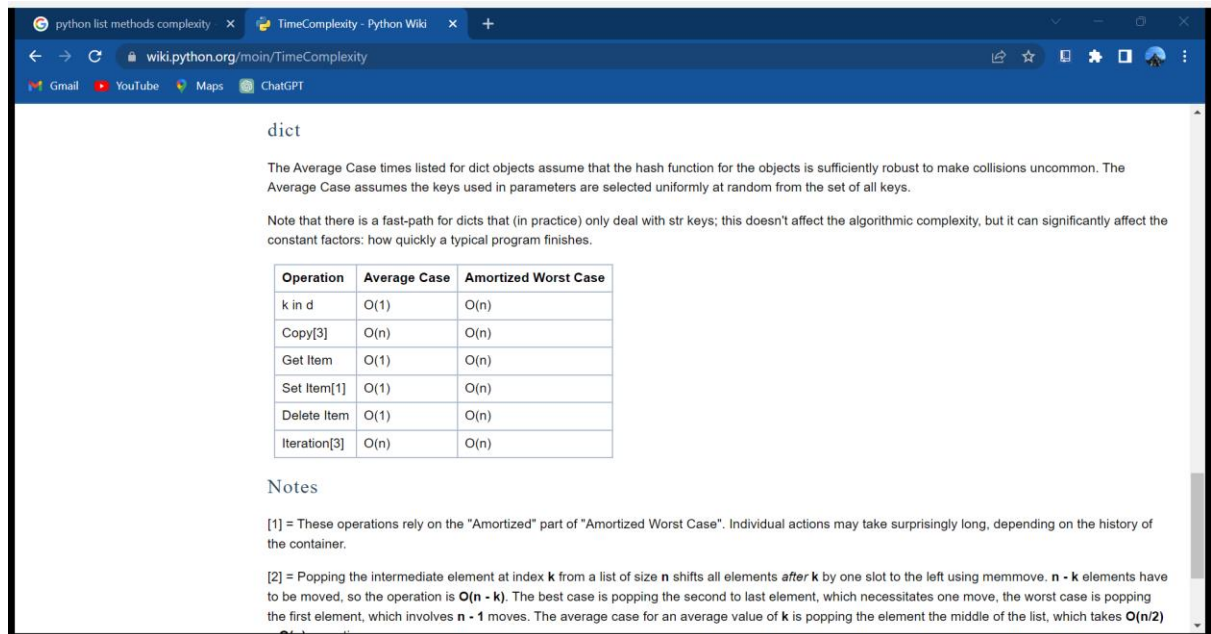
```

print('\n--- Oleh L200210137 ---')

```



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[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]	$O(n)$	$O(n)$

Notes

[1] = These operations rely on the "Amortized" part of "Amortized Worst Case". Individual actions may take surprisingly long, depending on the history of the container.

[2] = Popping the intermediate element at index k from a list of size n shifts all elements after k by one slot to the left using memmove. $n - k$ elements have to be moved, so the operation is $O(n - k)$. The best case is popping the second to last element, which necessitates one move, the worst case is popping the first element, which involves $n - 1$ moves. The average case for an average value of k is popping the element the middle of the list, which takes $O(n/2)$.

11. Notasi Big-O ($O(\cdot)$), notasi Big-Theta ($\Theta(\cdot)$), dan notasi Big-Omega ($\Omega(\cdot)$) adalah ketiga konsep yang berbeda dalam analisis kompleksitas algoritma.

a. **Notasi Big-O ($O(\cdot)$) :**

Notasi Big-O digunakan untuk memberikan batas atas pada pertumbuhan waktu eksekusi atau kompleksitas algoritma. Notasi Big-O memberikan batas atas pada kinerja algoritma, tetapi tidak memberikan informasi tentang kinerja terbaik atau rata-rata.

b. **Notasi Big-Theta ($\Theta(\cdot)$) :**

Notasi Big-Theta digunakan untuk memberikan batas atas dan batas bawah pada pertumbuhan waktu eksekusi atau kompleksitas algoritma. Algoritma tersebut memiliki kinerja yang stabil dan terikat oleh $f(n)$ baik pada kasus terburuk, terbaik, atau rata-rata.

c. **Notasi Big-Omega ($\Omega(\cdot)$) :**

Notasi Big-Omega digunakan untuk memberikan batas bawah pada pertumbuhan waktu eksekusi atau kompleksitas algoritma. Notasi Big-Omega memberikan batas bawah pada kinerja algoritma, tetapi tidak memberikan informasi tentang kinerja terbaik atau rata-rata.

12. Apa yang dimaksud dengan **amortized analysis** dalam analisis algoritma?

Amortized Analysis adalah metode untuk menganalisis kompleksitas waktu algoritma tertentu, terutama waktu atau memori yang dibutuhkan untuk eksekusi program. Tujuan dari amortized analysis adalah untuk melihat waktu eksekusi rata-rata dari sekumpulan operasi yang dilakukan oleh program.