Nama : Pasha Bhimasty

NIM : L200180123

Kelas : E

Modul 6

6.1 Menggabungkan dua list yang sudah urut

Latihan.py - D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Latihan.py (3.8.2)

File Edit Format Run Options Window Help

```
# 6.1 Menggabungkan dua list yang sudah urut
P = [2,8,15,23,37]
Q = [4,6,15,20]

def gabungkanDuaListUrut(A,B):
    la = len(A); lb = len(B);
    C = list()
    i = 0; j = 0;

    # Gabungkan keduanya sampai salah satu kosong
    while i < la and j < lb:
        if A[i] < B[j]:
            C.append(A[i])
            i += 1
        else:
            C.append(B[j])
            j += 1

    while i < la:
        C.append(A[i])
        i += 1

    while j < lb:
        C.append(B[j])
        j += 1

    return C

R = gabungkanDuaListUrut(P,Q)
print (R)
```

Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Latihan.py =======

======= RESTART: D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Latihan.py =======
[2, 4, 6, 8, 15, 15, 20, 23, 37]
```

6.2 Merge Sort

Latihan.py - D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Latihan.py (3.8.2)

File Edit Format Run Options Window Help

```
# 6.2 Merge Sort
def mergeSort(A):
    print("Membelah ",A)
    if len(A) > 1:
        mid = len(A) // 2
        separuhKiri = A[:mid]
        separuhKanan = A[mid:]

        mergeSort(separuhKiri)
        mergeSort(separuhKanan)

        # Di bawah ini merupakan proses penggabungan
        i = 0; j = 0; k = 0;
        while i < len(separuhKiri) and j < len(separuhKanan):
            if separuhKiri[i] < separuhKanan[j]:
                A[k] = separuhKiri[i]
                i = i + 1
            else:
                A[k] = separuhKanan[j]
                j = j + 1
            k = k + 1

        while i < len(separuhKiri):
            A[k] = separuhKiri[i]
            i = i + 1
            k = k + 1

        while j < len(separuhKanan):
            A[k] = separuhKanan[j]
            j = j + 1
            k = k + 1
    print ("Menggabungkan ",A)

alist = [54,26,93,17,77,31,44,55,20]
mergeSort(alist)

# 6.3 Quick Sort
def quickSort(A):
    quickSortBantu(A,0,len(A)-1)
```

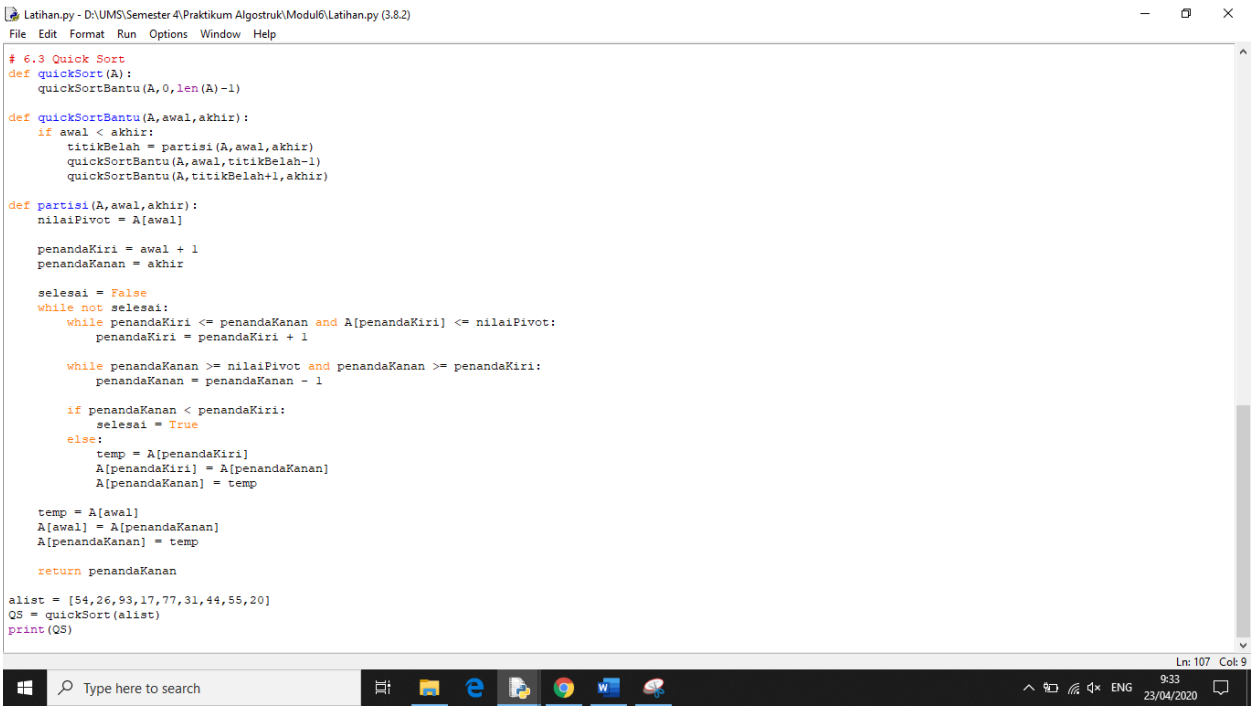
Python 3.8.2 Shell

File Edit Shell Debug Options Window Help

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======= RESTART: D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Latihan.py =======
Membelah  [54, 26, 93, 17, 77, 31, 44, 55, 20]
Membelah  [54, 26, 93, 17]
Membelah  [54, 26]
Membelah  [54]
Menggabungkan  [54]
Membelah  [26]
Menggabungkan  [26]
Menggabungkan  [26, 54]
Membelah  [93, 17]
Membelah  [93]
Menggabungkan  [93]
Membelah  [17]
Menggabungkan  [17]
Menggabungkan  [17, 93]
Menggabungkan  [17, 26, 54, 93]
Membelah  [77, 31, 44, 55, 20]
Membelah  [77, 31]
Membelah  [77]
Menggabungkan  [77]
Membelah  [31]
Menggabungkan  [31]
Menggabungkan  [31, 77]
Membelah  [44, 55, 20]
Membelah  [44]
Menggabungkan  [44]
Membelah  [55, 20]
Membelah  [55]
Menggabungkan  [55]
Membelah  [20]
Menggabungkan  [20]
Menggabungkan  [20, 55]
Menggabungkan  [20, 44, 55]
Menggabungkan  [20, 31, 44, 55, 77]
Menggabungkan  [17, 20, 26, 31, 44, 54, 55, 77, 93]
```

Ln: 39 Col: 0

6.3 Quick Sort



TUGAS

Nomor 1

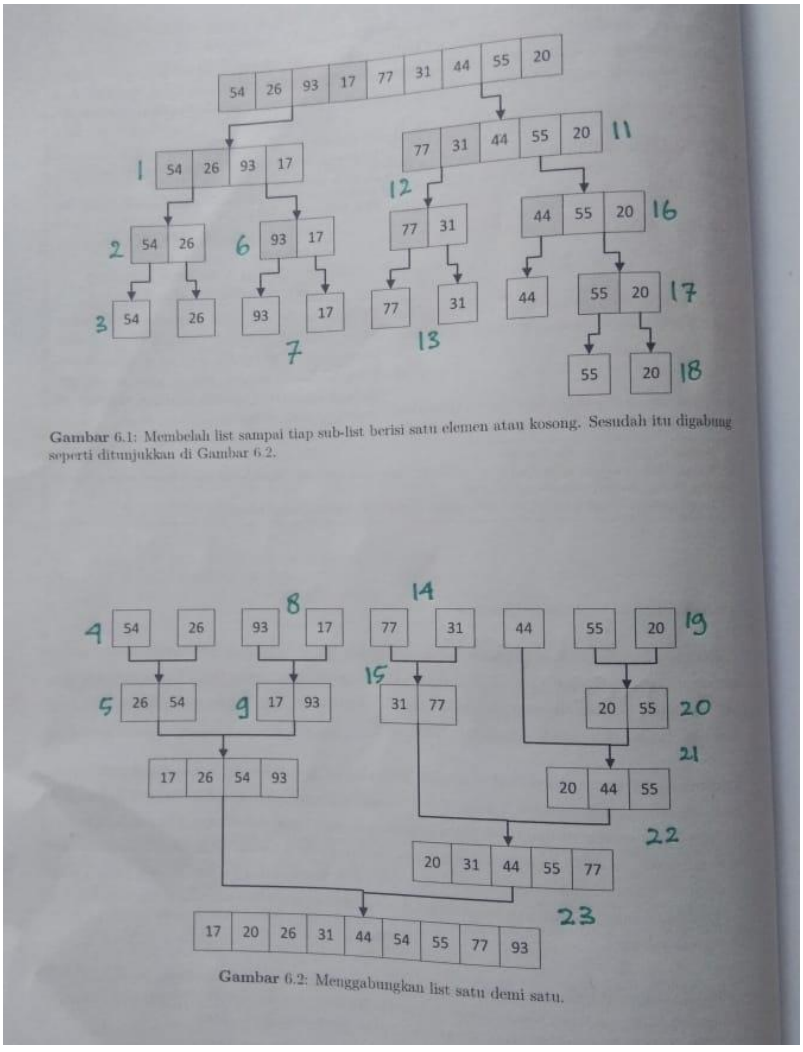
```
#======== no 1=========#
class mhs(object): #membuat class
    def __init__(self, nama, nim, kota, us): #metode pemanggil ketikan pemnuatan object terjadi
        self.nama = nama
        self.nim = nim
        self.kota = kota
        self.uang = us
    def __str__(self): #metode pemanggil ketika string akan di munculkan bersama pemanggilan onject
        x='Nama: '
        x+= self.nama + ', NIM: '
        x+= str(self.nim)+ ', Tempat tinggal: '
        x+= self.kota + ', Uang Saku: '
        x+= str(self.uang)
        return x
    def getNim(self): #metode pemanggil nim
        return self.nim
a0=mhs('Pasha', 123, 'Wonogiri', 240000)
a1=mhs('Damar', 126, 'Boyolali', 230000)
a2=mhs('Fira', 153, 'Surakarta', 250000)
a3=mhs('Anita', 185, 'Surakarta', 235000)
a4=mhs('Dika', 174, 'Boyolali', 240000)
a5=mhs('Rohana', 132, 'Brebes', 250000)
a6=mhs('Hanifah', 124, 'Klaten', 245000)
a7=mhs('Rema', 190, 'Wonogiri', 245000)
a8=mhs('Lika', 143, 'Klaten', 245000)
a9= mhs('Riri', 137, 'Karanganyar', 270000)
a10=mhs('Nunung',129,'Purwodadi',265000)

daftar = [a0,a1,a2,a3,a4,a5,a6,a7,a8,a9,a10] #list dari class mhsTIF
```

```
def mergesort(A): #untuk menghitung mergeshort

    if len(A) > 1:
        mid = len(A) // 2 #membelah list
        kiri = A[:mid] #membelah ke kiri
        kanan = A[mid:] #membelah ke kanan

        mergesort(kiri) #memanggil lebih lanjut mergeshort
        mergesort(kanan) #untuk separuh kiri dan separuh kanan

        i = 0 ; j = 0 ; k = 0
        while i < len (kiri) and j < len (kanan): #while lope ini
            if kiri[i].getNim() < kanan[j].getNim(): #jika loop ini
                A[k] = kiri[i] #menggabungkan kedua list
                i = i +1 #separuh kiri dan separuh kanan
            else:
                A[k] = kanan[j] #a samadengan kanan
                j = j +1 #maka kanan di tambah 1
            k = k+1 #dua list urut

        while i < len(kiri): #ketika i lebih kecil dari len kiri
            A[k]= kiri [i] #maka a samadengan kiri
            i= i+1 #kiri samadengan ditambah 1
            k = k +1 #k otomastis kiri dan di tambah 1

        while j < len(kanan): #ketika j lebih kecil dari kanan
            A[k]= kanan [j] #a samadengan kanan
            j= j+1 #maka kanan di tambah 1
            k = k +1 #k otomatis kanan dan di tambah 1
        return A
    #print 'menggabungkan' , A
```

```
def quickSort(a):
    quickSortbantu(a, 0, len(a)-1) #memanggil quickshort bantu
def quickSortbantu(a,awal,akhir):
    if awal <akhir:
        titikBelah = partisi (a, awal, akhir) #atur elemen dan dapatkan titik belah
        quickSortbantu(a, awal, titikBelah-1) #ini rekursi untuk belah sisi kiri
        quickSortbantu(a, titikBelah+1, akhir) #dan belah sisi kanan

def partisi(a,awal,akhir):
    nilaiPivot = a[awal].getNim() #nilai pivot di ambil dari elemen yg paling kiri disertai nim
    penandakiri = awal +1 #posisi awal penandakiri
    penandakanan = akhir #posisi awal penanda kanan
    selesai = False

    while not selesai: #loop untuk mengatur ulang posisi semua elemen
        while  penandakiri <= penandakanan and \
            a[penandakiri].getNim() <= nilaiPivot: #sampai ketemu suatu nilai yang
            penandakiri = penandakiri +1 #lebih besar dari nilai pivot
        while  a[penandakanan].getNim() >=nilaiPivot and penandakanan >= penandakiri:
            penandakanan = penandakanan -1
        if penandakanan < penandakiri: #kalau dua penanda sudah bersilangan
            selesai = True #selesai dan lanjut ke penempatan pivot
        else:
            a[penandakiri],a[penandakanan] = a[penandakanan], a[penandakiri]

    a[awal], a[penandakanan] = a[penandakanan], a[awal]
    return penandakanan #fungsi mengembalikan titik belah ke pemanggil

print ("\n")
print ("MERGE SORT")
mergesort(daftar)
for i in daftar:
    print (i) #untuk menampilkan list menggunakan mergesort dari daftar

print ("\n")
print ("QUICK SORT")
quickSort(daftar)
for i in daftar:
    print (i) #untuk menampilkan list menggunakan quicksort dari daftar
```

```
Python 3.8.2 Shell
File  Edit  Shell  Debug  Options  Window  Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: D:/UMS/Semester 4/Praktikum Algostruk/Modul6/Tugas.py ========


MERGE SORT
Nama: Pasha, NIM: 123, Tempat tinggal: Wonogiri, Uang Saku: 240000
Nama: Hanifah, NIM: 124, Tempat tinggal: Klaten, Uang Saku: 245000
Nama: Damar, NIM: 126, Tempat tinggal: Boyolali, Uang Saku: 230000
Nama: Nunung, NIM: 129, Tempat tinggal: Purwodadi, Uang Saku: 265000
Nama: Rohana, NIM: 132, Tempat tinggal: Brebes, Uang Saku: 250000
Nama: Riri, NIM: 137, Tempat tinggal: Karanganyar, Uang Saku: 270000
Nama: Lika, NIM: 143, Tempat tinggal: Klaten, Uang Saku: 245000
Nama: Fira, NIM: 153, Tempat tinggal: Surakarta, Uang Saku: 250000
Nama: Dika, NIM: 174, Tempat tinggal: Boyolali, Uang Saku: 240000
Nama: Anita, NIM: 185, Tempat tinggal: Surakarta, Uang Saku: 235000
Nama: Rema, NIM: 190, Tempat tinggal: Wonogiri, Uang Saku: 245000


QUICK SORT
Nama: Pasha, NIM: 123, Tempat tinggal: Wonogiri, Uang Saku: 240000
Nama: Hanifah, NIM: 124, Tempat tinggal: Klaten, Uang Saku: 245000
Nama: Damar, NIM: 126, Tempat tinggal: Boyolali, Uang Saku: 230000
Nama: Nunung, NIM: 129, Tempat tinggal: Purwodadi, Uang Saku: 265000
Nama: Rohana, NIM: 132, Tempat tinggal: Brebes, Uang Saku: 250000
Nama: Riri, NIM: 137, Tempat tinggal: Karanganyar, Uang Saku: 270000
Nama: Lika, NIM: 143, Tempat tinggal: Klaten, Uang Saku: 245000
Nama: Fira, NIM: 153, Tempat tinggal: Surakarta, Uang Saku: 250000
Nama: Dika, NIM: 174, Tempat tinggal: Boyolali, Uang Saku: 240000
Nama: Anita, NIM: 185, Tempat tinggal: Surakarta, Uang Saku: 235000
Nama: Rema, NIM: 190, Tempat tinggal: Wonogiri, Uang Saku: 245000
>>>
```

Nomor 2



Gambar 6.1: Membelah list sampai tiap sub-list berisi satu elemen atau kosong. Sesudah itu digabung seperti ditunjukkan di Gambar 6.2.

Gambar 6.2: Menggabungkan list satu demi satu.

Nomor 3

No2.py - C:\Users\pasha\Desktop\Modul6\No2.py (3.8.2)

File Edit Format Run Options Window Help

```
from time import time as detak
from random import shuffle as kocok
import time

def swap(A, p, q):
    tmp = A[p]
    A[p] = A[q]
    A[q] = tmp

def cariPosisiYangTerkecil(A, dariSini, sampaiSini):
    posisiYangTerkecil = dariSini
    for i in range(dariSini+1, sampaiSini):
        if A[i] < A[posisiYangTerkecil]:
            posisiYangTerkecil = i
    return posisiYangTerkecil

def bubbleSort(S):
    n = len(S)
    for i in range (n-1):
        for j in range (n-i-1):
            if S[j] > S[j+1]:
                swap(S,j,j+1)
    return S

def selectionSort(S):
    n = len(S)

    for i in range(n-1):
        indexKecil = cariPosisiYangTerkecil(S, i, n)
        if indexKecil != i:
            swap(S, i, indexKecil)
    return S
```

```
def insertionSort(S):
    n = len(S)
    for i in range(1, n):
        nilai = S[i]
        pos = i
        while pos > 0 and nilai < S[pos -1]:
            S[pos] = S[pos-1]
            pos = pos - 1
        S[pos] = nilai
    return S

def mergeSort(A):
    #print("Membelah      ",A)
    if len(A) > 1:
        mid = len(A) // 2
        separuhkiri = A[:mid]
        separuhkanan = A[mid:]

        mergeSort(separuhkiri)
        mergeSort(separuhkanan)

        i = 0;j=0;k=0
        while i < len(separuhkiri) and j < len(separuhkanan):
            if separuhkiri[i] < separuhkanan[j]:
                A[k] = separuhkiri[i]
                i = i + 1
            else:
                A[k] = separuhkanan[j]
                j = j + 1
            k=k+1

        while i < len(separuhkiri):
            A[k] = separuhkiri[i]
            i = i + 1
            k=k+1

        while j < len(separuhkanan):
            A[k] = separuhkanan[j]
            j = j + 1
            k=k+1
    #print("Menggabungkan",A)
```

```
def partisi(A, awal, akhir):
    nilaipivot = A[awal]

    penandakiri = awal + 1
    penandakanan = akhir

    selesai = False
    while not selesai:

        while penandakiri <= penandakanan and A[penandakiri] <= nilaipivot:
            penandakiri = penandakiri + 1

        while penandakanan >= penandakiri and A[penandakanan] >= nilaipivot:
            penandakanan = penandakanan - 1

        if penandakanan < penandakiri:
            selesai = True
        else:
            temp = A[penandakiri]
            A[penandakiri] = A[penandakanan]
            A[penandakanan] = temp

    temp = A[awal]
    A[awal] = A[penandakanan]
    A[penandakanan] = temp

    return penandakanan

def quickSortBantu(A, awal, akhir):
    if awal < akhir:
        titikBelah = partisi(A, awal, akhir)
        quickSortBantu(A, awal, titikBelah-1)
        quickSortBantu(A, titikBelah+1, akhir)

def quickSort(A):
    quickSortBantu (A, 0, len(A)-1)
```

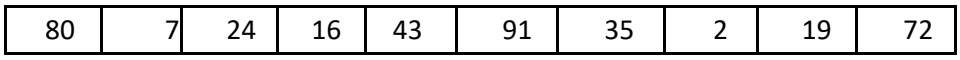
```
daftar = [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]

k =  [[i] for i in range(1, 6001)]
kocok(k)
u_bub = k[:]
u_sel = k[:]
u_ins = k[:]
u_mrg = k[:]
u_qck = k[:]

aw=detak();bubbleSort(u_bub);ak=detak();print("bubble: %g detik" %(ak-aw));
aw=detak();selectionSort(u_sel);ak=detak();print("selection: %g detik" %(ak-aw));
aw=detak();insertionSort(u_ins);ak=detak();print("insertion: %g detik" %(ak-aw));
aw=detak();mergeSort(u_mrg);ak=detak();print("merge: %g detik" %(ak-aw));
aw=detak();quickSort(u_qck);ak=detak();print("quick: %g detik" %(ak-aw));
```

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
================= RESTART: C:\Users\pasha\Desktop\Modul6\No2.py ================
bubble: 14.3271 detik
selection: 6.22341 detik
insertion: 6.6122 detik
merge: 0.0781183 detik
quick: 0.0780547 detik
>>>
```

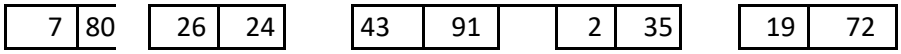
Nomor 4

1. Diberikan list L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72], gambarlah trace pengurutan untuk algoritma. a) Merge sort

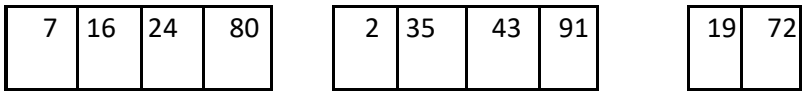
L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]



Proses 1

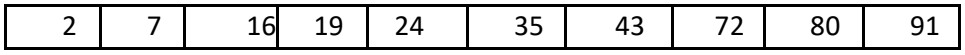


Proses 2



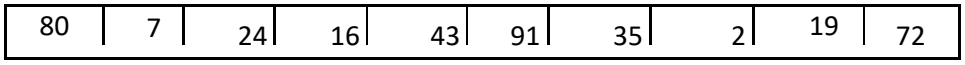
Proses 3

Proses 4



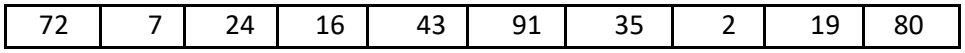
Quick sort

L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]



Pivot

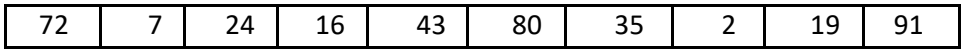
Low High Pivot

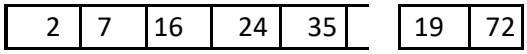


Low High Pivot

Low High

Pivot





b)

					Low				High
								Pivot	
72	7	24	16	43	19	35	2	80	91
					Low			High	

Pivot									
72	7	24	16	43	19	35	2	80	91
Low							High		

							Pivot		
2	7	24	16	43	19	35	72	80	91
Low							High		

Pivot									
2	7	24	16	43	19	35	72	80	91
Low						High			

	Pivot								
2	7	24	16	43	19	35	72	80	91
	Low					High			

		Pivot							
2	7	24	16	43	19	35	72	80	91
		Low				High			

		Pivot							
2	7	24	16	43	19	35	72	80	91
		Low			High				

					Pivot				
2	7	19	16	43	24	35	72	80	91
		Low			High				

					Pivot				
2	7	19	16	43	24	35	72	80	91
				Low	High				

				Pivot					
2	7	19	16	24	43	35	72	80	91

Low High

Pivot

2	7	19	16	24	43	35	72	80	91

Low High

Pivot

2	7	16	19	24	43	35	72	80	91

Low High

Pivot

2	7	16	19	24	43	35	72	80	91

Low High

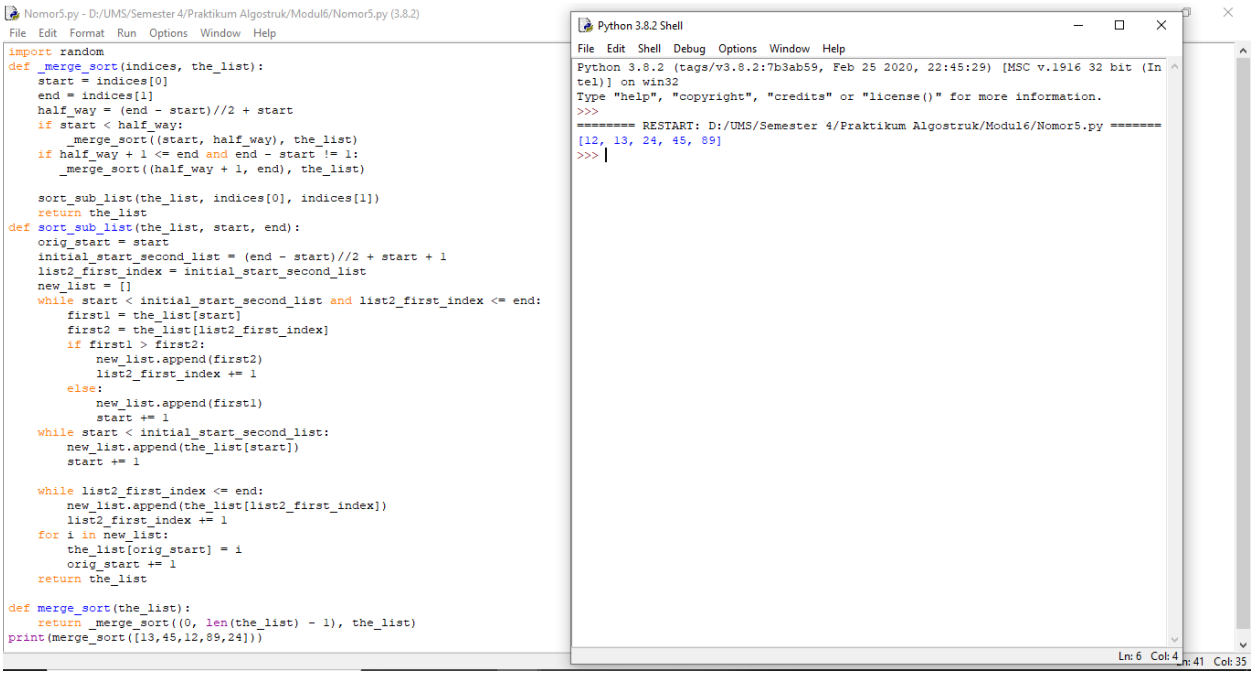
Pivot

2	7	16	19	24	35	43	72	80	91

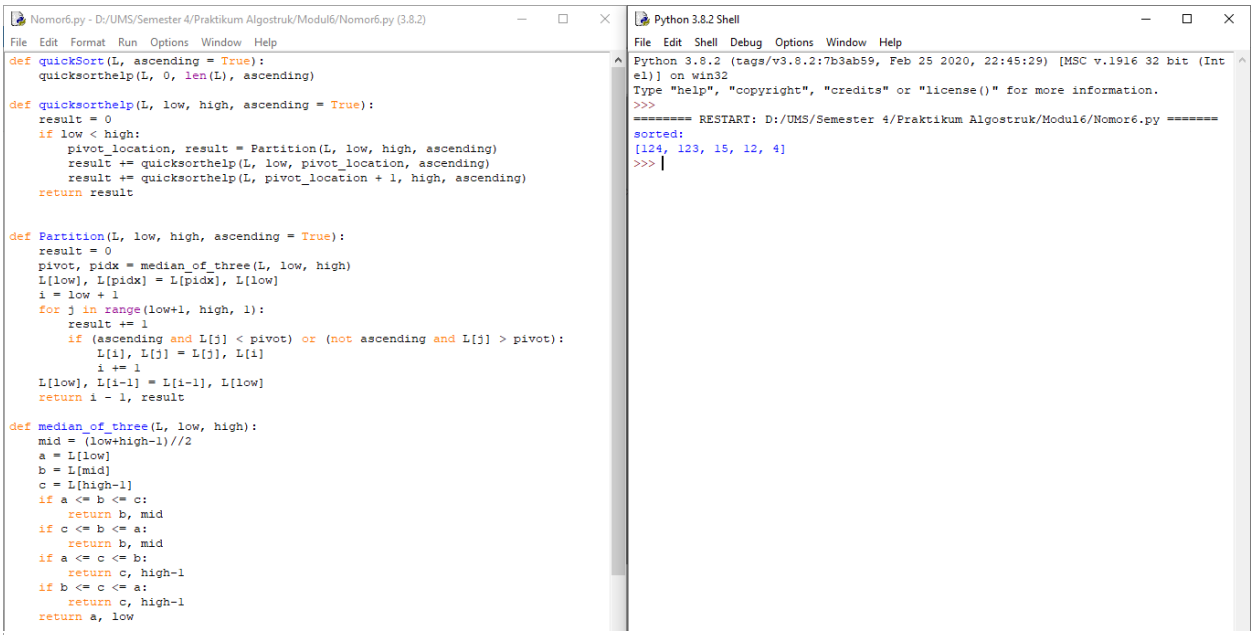
Low High

2	7	16	19	24	35	43	72	80	91

Nomor 5



Nomor 6


Nomor6.py - D:/UMS/Semester 4/Praktikum Algostruk/Modul6/Nomor6.py (3.8.2)
File Edit Format Run Options Window Help
def quickSort(L, ascending = True):
quicksorthelp(L, 0, len(L), ascending)
def quicksorthelp(L, low, high, ascending = True):
result = 0
if low < high:
pivot_location, result = Partition(L, low, high, ascending)
result += quicksorthelp(L, low, pivot_location, ascending)
result += quicksorthelp(L, pivot_location + 1, high, ascending)
return result
def Partition(L, low, high, ascending = True):
result = 0
pivot, pidx = median_of_three(L, low, high)
L[low], L[pidx] = L[pidx], L[low]
i = low + 1
for j in range(low+1, high, 1):
result += 1
if (ascending and L[j] < pivot) or (not ascending and L[j] > pivot):
L[i], L[j] = L[j], L[i]
i += 1
L[low], L[i-1] = L[i-1], L[low]
return i - 1, result
def median_of_three(L, low, high):
mid = (low+high-1)//2
a = L[low]
b = L[mid]
c = L[high-1]
if a <= b <= c:
return b, mid
if c <= b <= a:
return b, mid
if a <= c <= b:
return c, high-1
if b <= c <= a:
return c, high-1
return a, low
listel = list([12,4,15,124,123])
quickSort(listel, False) # descending order
print('sorted:')
print(listel)
Python 3.8.2 Shell
File Edit Shell Debug Options Window Help
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: D:/UMS/Semester 4/Praktikum Algostruk/Modul6/Nomor6.py =======
sorted:
[124, 123, 15, 12, 4]
>>>

Nomor 7

Nomor7.py - D:/UMS/Semester 4/Praktikum Algostruk/Modul6/Nomor7.py (3.8.2)

File Edit Format Run Options Window Help

```
from time import time as detak
from random import shuffle as kocok
import time
k = [i for i in range(1,6001)]
kocok(k)

def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        mergeSort(L)
        mergeSort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i+=1
            else:
                arr[k] = R[j]
                j+=1
            k+=1
        while i < len(L):
            arr[k] = L[i]
            i+=1
            k+=1
        while j < len(R):
            arr[k] = R[j]
            j+=1
            k+=1
def partition(arr,low,high):
    i = ( low-1 )
    pivot = arr[high]
    for j in range(low , high):
        if   arr[j] <= pivot:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]
    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )
```

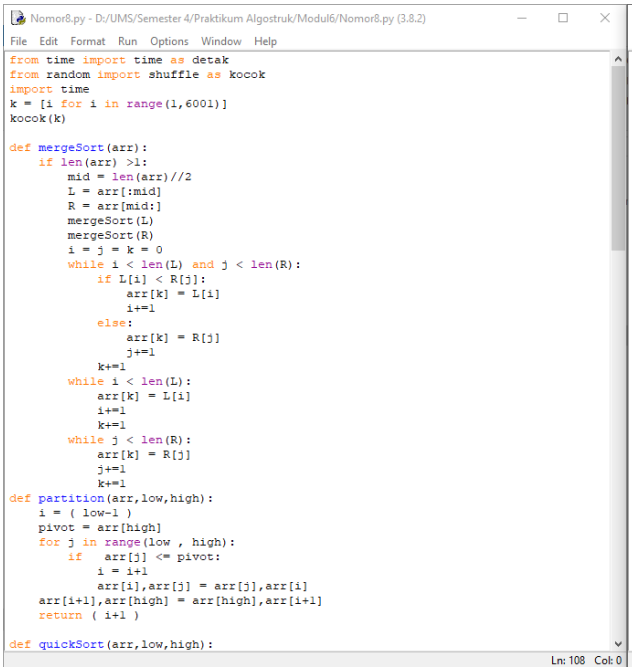
```
def quickSort(arr,low,high):
    if low < high:
        pi = partition(arr,low,high)
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

import random
def _merge_sort(indices, the_list):
    start = indices[0]
    end = indices[1]
    half_way = (end - start)//2 + start
    if start < half_way:
        _merge_sort((start, half_way), the_list)
    if half_way + 1 <= end and end - start != 1:
       _merge_sort((half_way + 1, end), the_list)

    sort_sub_list(the_list, indices[0], indices[1])
```

```
def sort_sub_list(the_list, start, end):
    orig_start = start
    initial_start_second_list = (end - start)//2 + start + 1
    list2_first_index = initial_start_second_list
    new_list = []
    while start < initial_start_second_list and list2_first_index <= end:
        first1 = the_list[start]
        first2 = the_list[list2_first_index]
        if first1 > first2:
            new_list.append(first2)
            list2_first_index += 1
        else:
            new_list.append(first1)
            start += 1
    while start < initial_start_second_list:
        new_list.append(the_list[start])
        start += 1

    while list2_first_index <= end:
        new_list.append(the_list[list2_first_index])
        list2_first_index += 1
    for i in new_list:
        the_list[orig_start] = i
        orig_start += 1

def merge_sort(the_list):
    return _merge_sort((0, len(the_list) - 1), the_list)

def quickSortMOD(L, ascending = True):
    quicksorthelp(L, 0, len(L), ascending)

def quicksorthelp(L, low, high, ascending = True):
    result = 0
    if low < high:
        pivot_location, result = Partition(L, low, high, ascending)
        result += quicksorthelp(L, low, pivot_location, ascending)
        result += quicksorthelp(L, pivot_location + 1, high, ascending)
    return result
```

```
def Partition(L, low, high, ascending = True):
    result = 0
    pivot, pidx = median_of_three(L, low, high)
    L[low], L[pidx] = L[pidx], L[low]
    i = low + 1
    for j in range(low+1, high, 1):
        result += 1
        if (ascending and L[j] < pivot) or (not ascending and L[j] > pivot):
            L[i], L[j] = L[j], L[i]
            i += 1
    L[low], L[i-1] = L[i-1], L[low]
    return i - 1, result

def median_of_three(L, low, high):
    mid = (low+high-1)//2
    a = L[low]
    b = L[mid]
    c = L[high-1]
    if a <= b <= c:
        return b, mid
    if c <= b <= a:
        return b, mid
    if a <= c <= b:
        return c, high-1
    if b <= c <= a:
        return c, high-1
    return a, low
mer = k[:]
qui = k[:]
mer2 = k[:]
qui2 = k[:]

aw=detak();mergeSort(mer);ak=detak();print('merge : %g detik' %(ak-aw));
aw=detak();quickSort(qui,0,len(qui)-1);ak=detak();print('quick : %g detik' %(ak-aw));
aw=detak();merge_sort(mer2);print('merge mod : %g detik' %(ak-aw));
aw=detak();quickSortMOD(qui2, False);print('quick mod : %g detik' %(ak-aw));
```

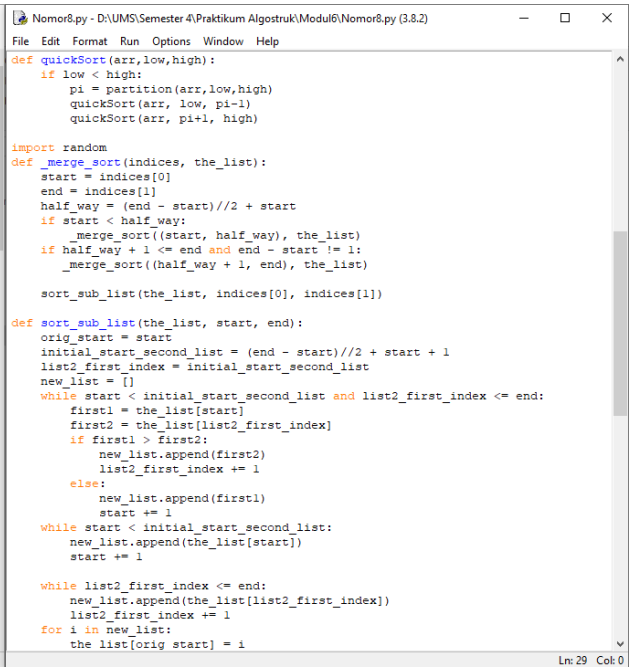
```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: D:/UMS/Semester 4/Praktikum Algostruk/Modul6/Nomor7.py =======
merge : 0.124985 detik
quick : 0.0625103 detik
merge mod : -0.0469129 detik
quick mod : -0.171863 detik
>>>
```

Nomor 8



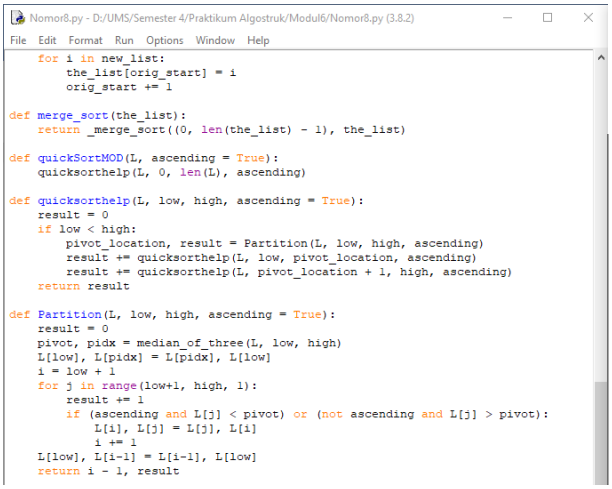
```
from time import time as detak
from random import shuffle as kocok
import time
k = [i for i in range(1,6001)]
kocok(k)

def mergeSort(arr):
    if len(arr) >1:
        mid = len(arr)//2
        L = arr[:mid]
        R = arr[mid:]
        mergeSort(L)
        mergeSort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                arr[k] = L[i]
                i+=1
            else:
                arr[k] = R[j]
                j+=1
            k+=1
        while i < len(L):
            arr[k] = L[i]
            i+=1
            k+=1
        while j < len(R):
            arr[k] = R[j]
            j+=1
            k+=1
def partition(arr,low,high):
    i = ( low-1 )
    pivot = arr[high]
    for j in range(low , high):
        if   arr[j] <= pivot:
            i = i+1
            arr[i],arr[j] = arr[j],arr[i]
    arr[i+1],arr[high] = arr[high],arr[i+1]
    return ( i+1 )

def quickSort(arr,low,high):
```

Ln: 108 Col: 0



```
def quickSort(arr,low,high):
    if low < high:
        pi = partition(arr,low,high)
        quickSort(arr, low, pi-1)
        quickSort(arr, pi+1, high)

import random
def _merge_sort(indices, the_list):
    start = indices[0]
    end = indices[1]
    half_way = (end - start)//2 + start
    if start < half_way:
        _merge_sort((start, half_way), the_list)
    if half_way + 1 <= end and end - start != 1:
        _merge_sort((half_way + 1, end), the_list)

    sort_sub_list(the_list, indices[0], indices[1])

def sort_sub_list(the_list, start, end):
    orig_start = start
    initial_start_second_list = (end - start)//2 + start + 1
    list2_first_index = initial_start_second_list
    new_list = []
    while start < initial_start_second_list and list2_first_index <= end:
        first1 = the_list[start]
        first2 = the_list[list2_first_index]
        if first1 > first2:
            new_list.append(first2)
            list2_first_index += 1
        else:
            new_list.append(first1)
            start += 1
    while start < initial_start_second_list:
        new_list.append(the_list[start])
        start += 1

    while list2_first_index <= end:
        new_list.append(the_list[list2_first_index])
        list2_first_index += 1
    for i in new_list:
        the_list[orig_start] = i
```

Ln: 29 Col: 0



```
    for i in new_list:
        the_list[orig_start] = i
        orig_start += 1

def merge_sort(the_list):
    return _merge_sort((0, len(the_list) - 1), the_list)

def quickSortMOD(L, ascending = True):
    quicksorthelp(L, 0, len(L), ascending)

def quicksorthelp(L, low, high, ascending = True):
    result = 0
    if low < high:
        pivot_location, result = Partition(L, low, high, ascending)
        result += quicksorthelp(L, low, pivot_location, ascending)
        result += quicksorthelp(L, pivot_location + 1, high, ascending)
    return result

def Partition(L, low, high, ascending = True):
    result = 0
    pivot, pidx = median_of_three(L, low, high)
    L[low], L[pidx] = L[pidx], L[low]
    i = low + 1
    for j in range(low+1, high, 1):
        result += 1
        if (ascending and L[j] < pivot) or (not ascending and L[j] > pivot):
            L[i], L[j] = L[j], L[i]
            i += 1
    L[low], L[i-1] = L[i-1], L[low]
    return i - 1, result
```

```
def median_of_three(L, low, high):
    mid = (low+high-1)//2
    a = L[low]
    b = L[mid]
    c = L[high-1]
    if a <= b <= c:
        return b, mid
    if c <= b <= a:
        return b, mid
    if a <= c <= b:
        return c, high-1
    if b <= c <= a:
        return c, high-1
    return a, low
mer = k[:]
qui = k[:]
mer2 = k[:]
qui2 = k[:]

aw=detak();mergeSort(mer);ak=detak();print('merge : %g detik' %(ak-aw));
aw=detak();quickSort(qui,0,len(qui)-1);ak=detak();print('quick : %g detik' %(ak-aw));
aw=detak();merge_sort(mer2);print('merge mod : %g detik' %(ak-aw));
aw=detak();quickSortMOD(qui2, False);print('quick mod : %g detik' %(ak-aw));
```

```
Python 3.8.2 (tags/v3.8.2:7b3ab59, Feb 25 2020, 22:45:29) [MSC v.1916 32 bit (In
tel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
======== RESTART: D:\UMS\Semester 4\Praktikum Algostruk\Modul6\Nomor8.py =======
merge : 0.124986 detik
quick : 0.0468726 detik
merge mod : 0 detik
quick mod : -0.125049 detik
>>>
```