

Nama : Zulfa fajrul Falah
NIM : L200170149
Kelas : D

LAPORAN PRAKTIKUM ALGORITMA DAN STRUKTUR DATA MODUL 6

1. Ubahlah kode mergeSort dan quickSort di atas agar bisa mengurutkan list yang berisi object-object mhsTIF yang sudah kamu buat di Modul 2.

```
class mhs(object): #membuat class
    def __init__(self, nama, nim, kota, us): #metode pemanggil ketika 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('Falah', 10, 'Boyolali', 240000)
a1=mhs('sulu', 51, 'Sragen', 230000)
a2=mhs('sule', 2, 'Surakarta', 250000)
a3=mhs('cocol', 18, 'Surakarta', 235000)
a4=mhs('dodol', 4, 'Boyolali', 240000)
a5=mhs('dims', 31, 'Brebes', 250000)
a6=mhs('Doso', 13, 'Klaten', 245000)
a7=mhs('galowi', 5, 'Wonogiri', 245000)
a8=mhs('Janto', 23, 'Klaten', 245000)
a9= mhs('hiawa', 64, 'Karanganyar', 270000)
a10=mhs('sitik',29,'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
```

```

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 otomatis 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:

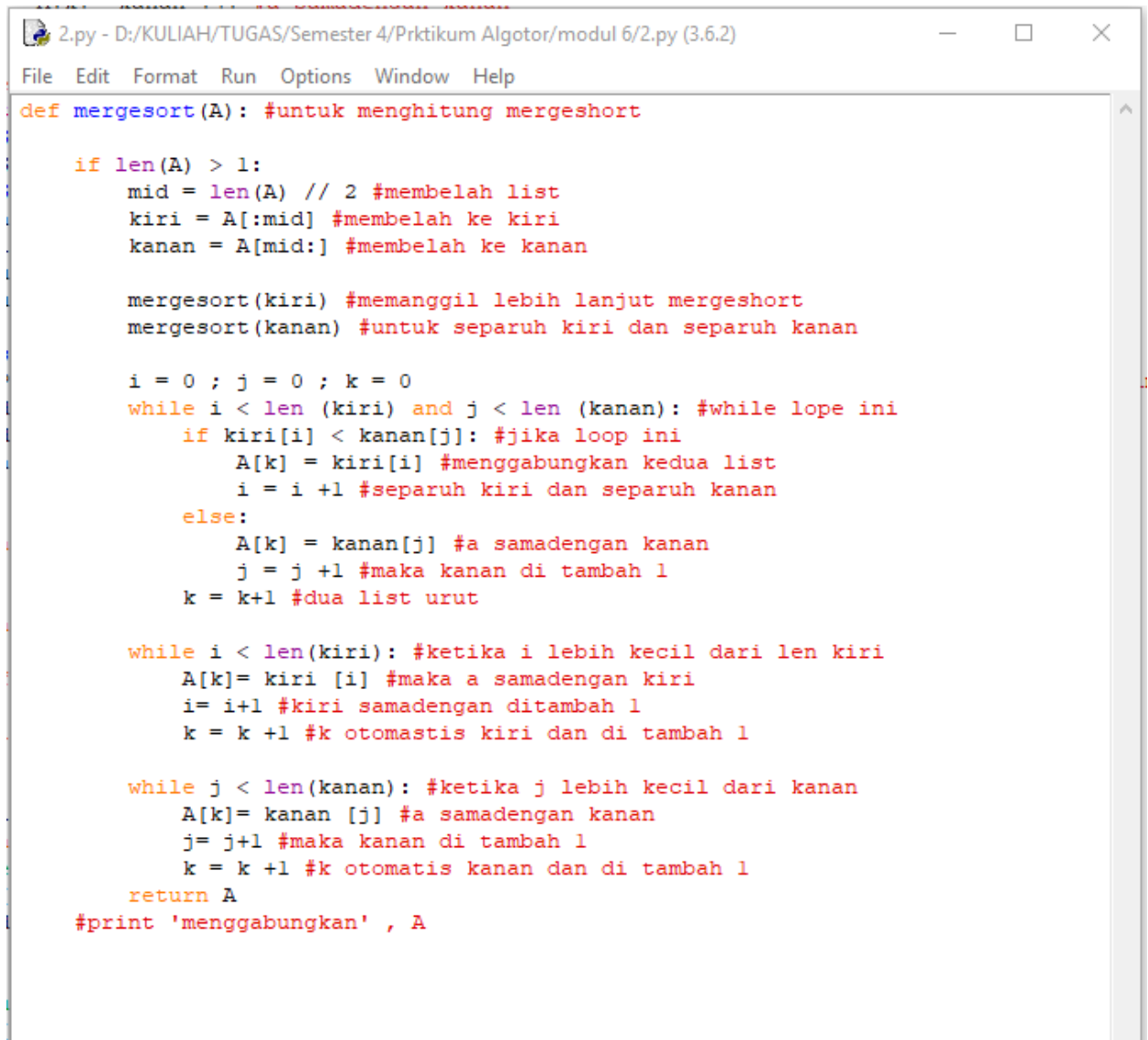
            while a[penandakanan].getNim() >=nilaiPivot and penandakanan >= penandakiri:
                penandakanan = penandakanan -1
            if penandakanan < penandakiri: #kalau dua penanda sudah bersilangan
                selesai = True #selesai dan lanjut ke penempatan pivot
            else:
                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 ("merge sort")
mergesort(daftar)
for i in daftar:
    print (i) #untuk menampilkan list menggunakan mergesort dari daftar

print ('quick sort')
quickSort(daftar)
for i in daftar:
    print (i) #untuk menampilkan list menggunakan quicksort dari daftar

```

2. Memakai bolpen merah atau biru, tandai dan beri nomor urut eksekusi proses pada Gambar 6.1 dan 6.2.



```
2.py - D:/KULIAH/TUGAS/Semester 4/Prktikum Algoritma/modul 6/2.py (3.6.2)
File Edit Format Run Options Window Help

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 loop ini
        if kiri[i] < kanan[j]: #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 otomatis 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
```

3. Uji kecepatan. Ujilah mergeSort dan quickSort.

```
from time import time as detik
from random import shuffle as kocok
k= range(6000)
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);
```

Hasil:

```
bubble: 5.114 detik
selection: 1.552 detik
insertion: 2.14 detik
merge: 0.0419998 detik
quick: 0.0190001 detik
>>> |
```

4. None

5. Nomor 5

```
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])
    return the_list

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
    return the_list

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

print(merge_sort([13,45,12]))
```

Nomor 6

```
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)
```

Nomor 7

```
from time import time as detik
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

```

```

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));

```

Nomor 8

```
from time import time as detik
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));

```