

## 6.4 Soal-soal untuk Mahasiswa

Sebelum mengerjakan soal-soal dibawah, kerjakan dulu latihan-latihan diatas.

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

```
print('\n--- Oleh L200220037 ---')

print ( "Nomor 1")

class Mahasiswa(object):
    def __init__ (self,nim) :
        self.nim = nim

a1= "L200170156"
a2= "L200170152"
a3= "L200170155"
a4= "L200170147"
a5= "L200170143"

Daftar = [a1,a2,a3,a4,a5]
def mergeSort(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

mergeSort(Daftar)
print("Menggunakan Merge Sort : \n",Daftar)

def quickSort(A):
```

```

    quickSortBantu(A,0,len(A) - 1)

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

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 A[penandaKanan] >= nilaiPivot and \
            penandaKanan >= penandaKiri :
            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

quickSort(Daftar)
print("Menggunakan Quick Sort : \n",Daftar)

```

Output:

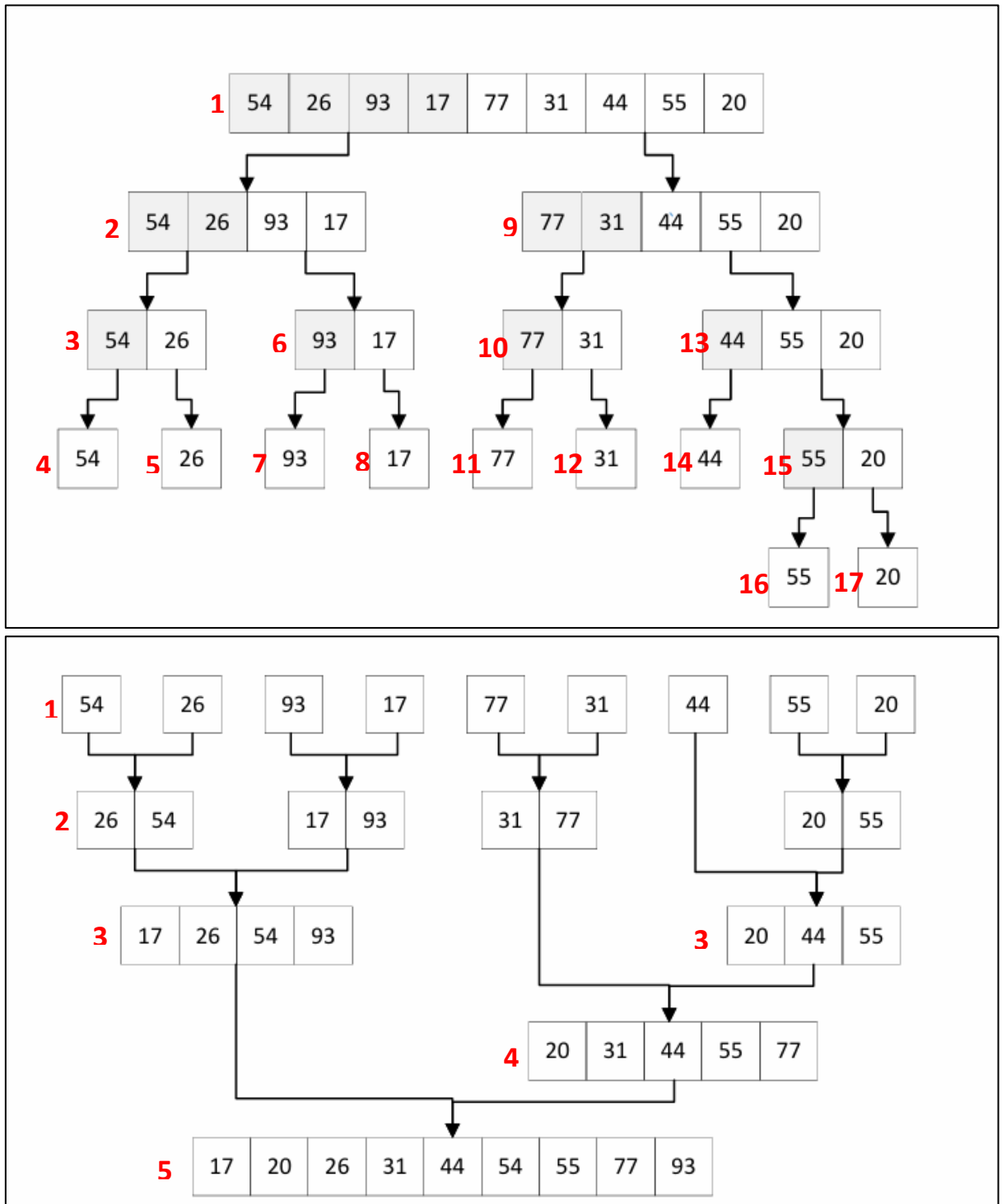
```

PS D:\KULIAH\MATERI\SEMESTER 4\PRAKTIKUM ALGORITMA STRUKTUR DATA\modul6> & C:/
cal/Microsoft/WindowsApps/python3.10.exe "d:/KULIAH/MATERI/SEMESTER 4/PRAKTIK
DATA/modul6/1.py"

--- Oleh L200220037 ---
Nomor 1
Menggunakan Merge Sort :
['L200170143', 'L200170147', 'L200170152', 'L200170155', 'L200170156']
Menggunakan Quick Sort :
['L200170143', 'L200170147', 'L200170152', 'L200170155', 'L200170156']
PS D:\KULIAH\MATERI\SEMESTER 4\PRAKTIKUM ALGORITMA STRUKTUR DATA\modul6> 

```

2. Memakai bolpen merah atau biru, tandai dan beri nomer urut eksekusi proses pada Gambar 6.1 dan 6.2, dengan mengacu pada output di halaman 59.



3. Uji kecepatan. Ujilah mergeSort dan quickSort diatas (bersama metode sort yang kamu pelajari sebelumnya) dengan kode dibawah ini.

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

```

k = range(6000)
kocok(k)
u_bub = k[:] ##
u_sel = k[:] ## Deep copy.
u_ins = k[:] ## Jangan lupa [:]-nya!
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) );

```

Tunjukkan hasil ujinya keasisten praktikum.

```

print('\n--- Oleh L200220037 ---')

print ("Nomor 3")
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 bubbleSort(A):
    n = len(A)
    for i in range(n-1):
        for j in range (n-i-1):
            if A[j] > A[j+1]:
                swap(A,j,j+1)

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

def selectionSort(A):
    n = len(A)
    for i in range(n-1):
        indexKecil = cariPosisiYangTerkecil(A, i, n)
        if indexKecil != i:
            swap(A, i, indexKecil)

```

```

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

def mergeSort(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

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

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

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 A[penandaKanan] >= nilaiPivot and \
            penandaKanan >= penandaKiri :
            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

k=[]
for i in range(1, 6001):
    k.append(i)
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));

```

Output:

```

PS D:\KULIAH\MATERI\SEMESTER 4\PRAKTIKUM ALGORITMA STR
rosoft/windowsApps/python3.10.exe "d:/KULIAH/MATERI/SE
py"

--- Oleh L200220037 ---
Nomor 3
bubble : 1.34832 detik
selection: 1.10624 detik
insertion : 1.4981 detik
merge: 0.0213094 detik
quick : 0.0129879 detik

```

4. Diberikan list  $L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]$ , gambarlah trace pengurutan untuk algoritma

a) Mergesort

80	7	24	16	43	91	35	2	19	72
----	---	----	----	----	----	----	---	----	----

Proses 1

7	80		26	24		43	91		2	35		19	72
---	----	--	----	----	--	----	----	--	---	----	--	----	----

Proses 2

7	16	24	80		2	35	43	91			19	72
---	----	----	----	--	---	----	----	----	--	--	----	----

Proses 3

2	7	16	24	35	43	80	91			19	72
---	---	----	----	----	----	----	----	--	--	----	----

Proses 4

2	7	16	19	24	35	43	72	80	91
---	---	----	----	----	----	----	----	----	----

b) Quicksort

pivot

80	7	24	16	43	91	35	2	19	72
----	---	----	----	----	----	----	---	----	----

low

high

pivot

72	7	24	16	43	91	35	2	19	80
----	---	----	----	----	----	----	---	----	----

low

high

pivot

72	7	24	16	43	91	35	2	19	80
----	---	----	----	----	----	----	---	----	----

low

high

pivot

72	7	24	16	43	80	35	2	19	91
----	---	----	----	----	----	----	---	----	----

low

high

pivot

72	7	24	16	43	19	35	2	80	91
----	---	----	----	----	----	----	---	----	----

low

high

Soal-soal dibawah ini sedikit lebih sulit. Kerjakanlah dirumah.

5. Tingkatkan efisiensi program mergeSort dengan tidak memakai operator slice (seperti `A[:mid]` dan `A[mid:]`), dan lalu mem-pass index awal dan index akhir bersama listnya saat kita memanggil mergeSort secara rekursif. Kamu akan perlu memisah fungsi mergeSort itu menjadi beberapa fungsi, mirip halnya dengan apa yang dilakukan algoritma quickSort.

```
print('\n--- Oleh L200220037 ---')

print ("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]))
```



Output:

```
rosoft/windowsApps/python3.10.exe
py"

--- Oleh L200220037 ---
Nomor 5
[12, 13, 45]
```

6. Apakah kita bisa meningkatkan efisiensi program quicksort dengan memakai metode median-dari-tiga untuk memilih pivotnya? Ubahlah kodenya dan ujilah.

```
print('\n--- Oleh L200220037 ---')

print("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

liste1 = list([12,4,15,124,123])

quickSort(liste1, False) # descending order
print('sorted:')
print(liste1)

```

Output:

```

rosoft/WindowsApps/python3.10.exe "d
py"

--- Oleh L200220037 ---
Nomor 6
sorted:
[124, 123, 15, 12, 4]

```

7. Uji-kecepatan keduanya dan perbandingkan juga dengan kode awalnya.

```

print('\n--- Oleh L200220037 ---')

print("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

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));
```

Output:

```
rosoft/WindowsApps/python3.10.exe "d:/KULIAH/
py"

--- Oleh L200220037 ---
Nomor 7
merge : 0.0500157 detik
quick : 0.0274694 detik
merge mod : 0 detik
quick mod : -0.0557785 detik
```

8. Buatlah versi linked-list untuk program mergeSort diatas.

```
print('\n--- Oleh L200220037 ---')

print("Nomor 8")
class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

class LinkedList:
    def __init__(self):
        self.head = None

    def appendList(self, data):
        node = Node(data)
        if self.head == None:
            self.head = node
        else:
            curr = self.head
            while curr.next != None:
                curr = curr.next
            curr.next = node

    def appendSorted(self, data):
        node = Node(data)
        curr = self.head
        prev = None

        while curr is not None and curr.data < data:
            prev = curr
            curr = curr.next

        if prev == None:
```

```

        self.head = node
    else:
        prev.next = node

    node.next = curr

def printList(self):
    curr = self.head
    while curr != None:
        print ("%d"%curr.data),
        curr = curr.next
def mergeSorted(self, list1, list2):
    if list1 is None:
        return list2
    if list2 is None:
        return list1

    if list1.data < list2.data:
        temp = list1
        temp.next = self.mergeSorted(list1.next, list2)
    else:
        temp = list2
        temp.next = self.mergeSorted(list1, list2.next)
    return temp

list1 = LinkedList()
list1.appendSorted(13)
list1.appendSorted(12)
list1.appendSorted(3)
list1.appendSorted(16)
list1.appendSorted(7)

print("List 1 :"),
list1.printList()

list2 = LinkedList()
list2.appendSorted(9)
list2.appendSorted(10)
list2.appendSorted(1)

print("List 2 :"),
list2.printList()

list3 = LinkedList()
list3.head = list3.mergeSorted(list1.head, list2.head)

print("Merged List :"),
list3.printList()

```

Output:

--- Oleh L200220037 ---

Nomor 8

List 1 :

3

7

12

13

16

List 2 :

1

9

10

Merged List :

1

3

7

9

10

12

13

16

PS D:\KULIAH\MATERI\SEMESTER 4\PRAKTIKUM ALGORITMA ST