Nama : Nadya Ayu Widya

NIM : L200180099

Kelas : D

Tugas Praktikum Modul 6

Nomor 1. Berikut program:

```
class MhsTIF():
    def __init__(self,nama,NIM,kota,us):
        self.nama = nama
        self.NIM = NIM
        self.kota = kota
        self.us = us

c0 = MhsTIF('Wulan', 'L200180091', 'Sukoharjo', 240000)
c1 = MhsTIF('Fandit', 'L200180092', 'Sragen', 230000)
c2 = MhsTIF('Abid', 'L200180093', 'Sukoharjo', 250000)
c3 = MhsTIF('Elsa', 'L200180094', 'Sukoharjo', 230000)
c4 = MhsTIF('Ayud', 'L200180095', 'Boyolali', 240000)
c5 = MhsTIF('Nopal', 'L200180096', 'Salatiga', 230000)
c6 = MhsTIF('Chandika', 'L200180097', 'Klaten', 245000)
c7 = MhsTIF('Berlin', 'L200180098', 'Wonogiri', 245000)
c8 = MhsTIF('Nayu', 'L200180099', 'Klaten', 245000)
c9 = MhsTIF('Rayhan', 'L200180100', 'Karanganyar', 270000)
c10 = MhsTIF('Irul', 'L200180101', 'Purwodadi', 265000)

Daftar = [c0, c1, c2, c3, c4, c5, c6, c7, c8, c9, c10]

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].NIM < separuhKanan[j].NIM :
                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
```

```
        while j < len(separuhKanan):
            A[k] = separuhKanan[j]
            j = j+1
            k = k+1
mergeSort(Daftar)
print('Menggunakan Merge Sort:')
for i in Daftar:
    print(i.NIM)

def partisi(A,awal,akhir):
    i = awal - 1
    nilaiPivot = A[akhir].us
    for j in range(awal, akhir):
        if A[j].us <= nilaiPivot:
            i = i + 1
            A[i].us, A[j].us = A[j].us, A[i].us
    A[i + 1].us, A[akhir].us = A[akhir].us, A[i + 1].us
    return (i + 1)

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

print("####################Nomor 1########################")
quickSort(Daftar, 0, len(Daftar)-1)
print('Menggunakan Quick Sort:')
for i in Daftar:
    print(i.us)
```

Berikut output:

```
Python 3.7.7 (tags/v3.7.7:d7c567b08f, Mar 10 2020, 10:41:24) [MSC v.1900 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
============ RESTART: D:/Semester 4/Praktikum ASD/Modul 6/Tugas.py ============
Menggunakan Merge Sort:
L200180091
L200180092
L200180093
L200180094
L200180095
L200180096
L200180097
L200180098
L200180099
L200180100
L200180101
####################Nomor 1######################
Menggunakan Quick Sort:
230000
230000
230000
240000
240000
245000
245000
245000
250000
265000
270000
>>>
```

Nomor 3. Berikut program:

```
from time import time as detak
from random import shuffle as kocok
import time
def swap(A,p,q):
    tmp = A[p]
    A[q]= 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]
```

```
        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 partisi(A,awal,akhir):
    i = awal - 1
    nilaiPivot = A[akhir]
    for j in range(awal, akhir):
        if A[j] <= nilaiPivot:
            i = i + 1
            A[i], A[j] = A[j], A[i]
    A[i + 1], A[akhir] = A[akhir], A[i + 1]
    return (i + 1)

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

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-a
aw = detak();mergeSort(u_mrg);ak=detak();print("merge: %g detik" %(ak-aw));
aw = detak();quickSort(u_qck, 0, len(u_qck)- 1);ak=detak();print("quick : %g det
```

Berikut output:

```
bubble : 1.84715 detik
selection: 1.19337 detik
insertion : 2.06508 detik
merge: 0.0274327 detik
quick : 0.00558305 detik
>>>
```

Nomor 4. Berikut program:

```
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 mergeSort(the_list):
    return _merge_sort((0, len(the_list) - 1), the_list)
```

```
print('Merge Sort:\n',mergeSort([61,55,16]))
```

Berikut output:

```
Merge Sort:
 [16, 55, 61]
>>>
```

Nomor 6. Berikut program:

```
def quickSort(A, ascending = True):
    quicksortBantu(A, 0, len(A), ascending)

def quicksortBantu(A, awal, akhir, ascending = True):
    hasil = 0
    if awal < akhir:
        lokasi_privot, hasil = Partisi(A, awal, akhir, ascending)
        hasil += quicksortBantu(A, awal, lokasi_privot, ascending)
        hasil += quicksortBantu(A, lokasi_privot + 1, akhir, ascending)
    return hasil


def Partisi(A, awal, akhir, ascending = True):
    hasil = 0
    pivot, pidx = median_of_three(A, awal, akhir)
    A[awal], A[pidx] = A[pidx], A[awal]
    i = awal + 1
    for j in range(awal+1, akhir, 1):
        hasil += 1
        if (ascending and A[j] < pivot) or (not ascending and A[j] > pivot):
            A[i], A[j] = A[j], A[i]
            i += 1
    A[awal], A[i-1] = A[i-1], A[awal]
    return i - 1, hasil
```

```
def median_of_three(A, awal, akhir):
    mid = (awal+akhir-1)//2
    a = A[awal]
    b = A[mid]
    c = A[akhir-1]
    if a <= b <= c:
        return b, mid
    if c <= b <= a:
        return b, mid
    if a <= c <= b:
        return c, akhir-1
    if b <= c <= a:
        return c, akhir-1
    return a, awal

List1 = list([15,3,18,132,133])

quickSort(List1, False)  # descending order
print('Sorted:')
print(List1)
```

Berikut output:

```
Sorted:
[133, 132, 18, 15, 3]
>>>
```

Nomor 7. Berikut program:

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

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 partisi(A,awal,akhir):
    i = awal - 1
    nilaiPivot = A[akhir]
    for j in range(awal, akhir):
        if A[j] <= nilaiPivot:
            i = i + 1
            A[i], A[j] = A[j], A[i]
    A[i + 1], A[akhir] = A[akhir], A[i + 1]
    return (i + 1)
def quickSort(A,awal,akhir):
    if awal < akhir :
        titikBelah = partisi (A, awal, akhir)
        quickSort(A,awal,titikBelah - 1)
        quickSort(A,titikBelah + 1, akhir)

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 quickSortBanding(A, ascending = True):
    quicksortBantu(A, 0, len(A), ascending)

def quicksortBantu(A, awal, akhir, ascending = True):
    hasil = 0
    if awal < akhir:
        lokasi_privot, hasil = Partition(A, awal, akhir, ascending)
        hasil += quicksortBantu(A, awal, lokasi_privot, ascending)
        hasil += quicksortBantu(A, lokasi_privot + 1, akhir, ascending)
    return hasil


def median_of_three(A, awal, akhir):
    mid = (awal+akhir-1)//2
    a = A[awal]
    b = A[mid]
    c = A[akhir-1]
    if a <= b <= c:
        return b, mid
    if c <= b <= a:
        return b, mid
    if a <= c <= b:
        return c, akhir-1
    if b <= c <= a:
        return c, akhir-1
    return a, awal

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

u_mrg = k[:]
u_qck = k[:]
u_mrg2 = k[:]
u_qck2 = k[:]


aw = detak();mergeSort(u_mrg);ak=detak();
print("Merge Sort: %g detik" %(ak-aw));
aw = detak();quickSort(u_qck, 0, len(u_qck)- 1);ak=detak();
print("Quick Sort : %g detik" %(ak-aw));
aw = detak();merge_sort(u_mrg2);
print('Merge Sort 2 : %g detik' % (ak - aw));
aw = detak();quickSortBanding(u_qck2, False);
print('Quick Sort 2 : %g detik' % (ak - aw));
```

Berikut output:

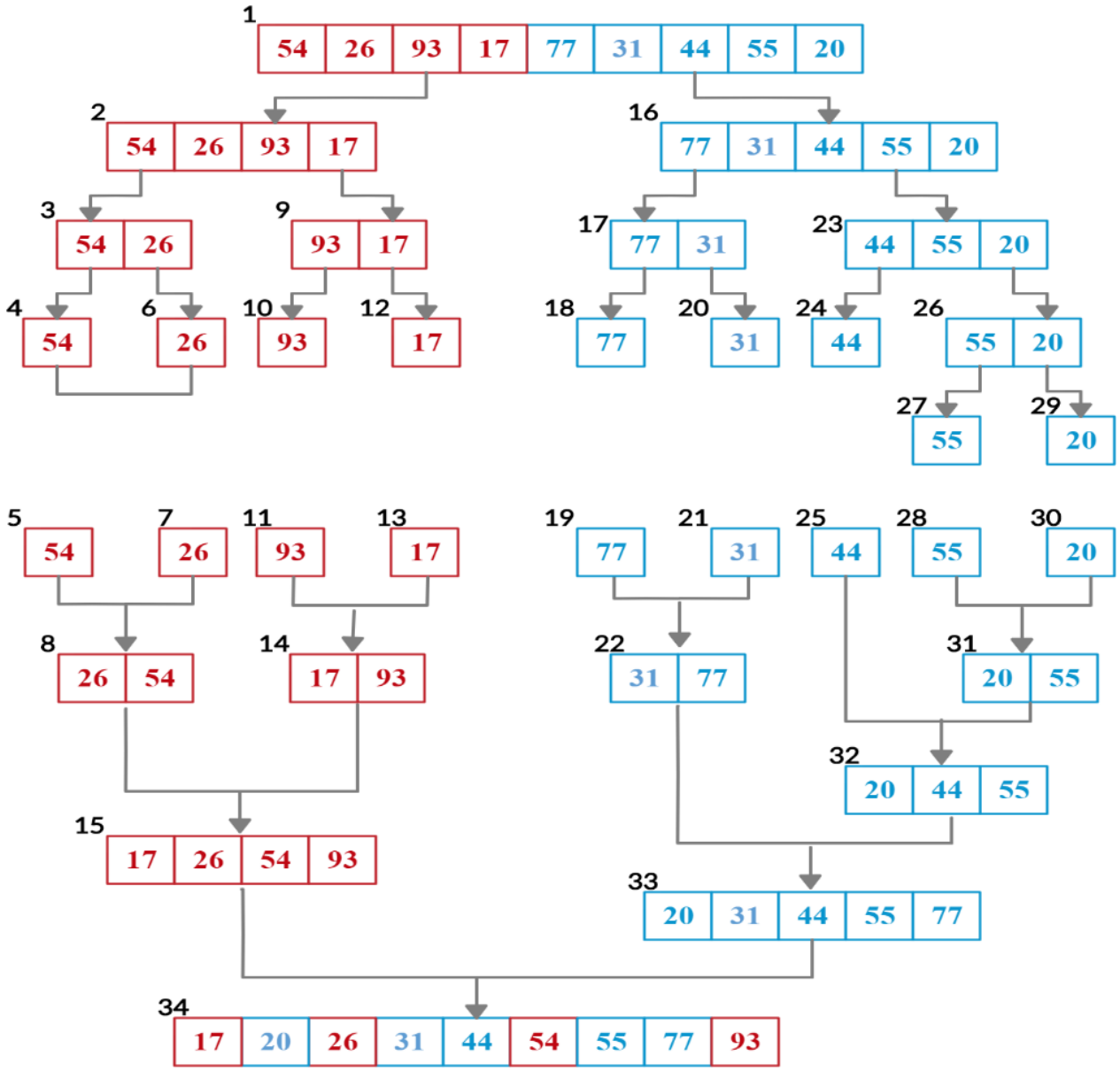
```
Merge Sort: 0.0644174 detik
Quick Sort : 0.0397177 detik
Merge Sort 2 : -0.00881028 detik
Quick Sort 2 : -0.0457993 detik
>>>
```

Nomor 8. Berikut program:

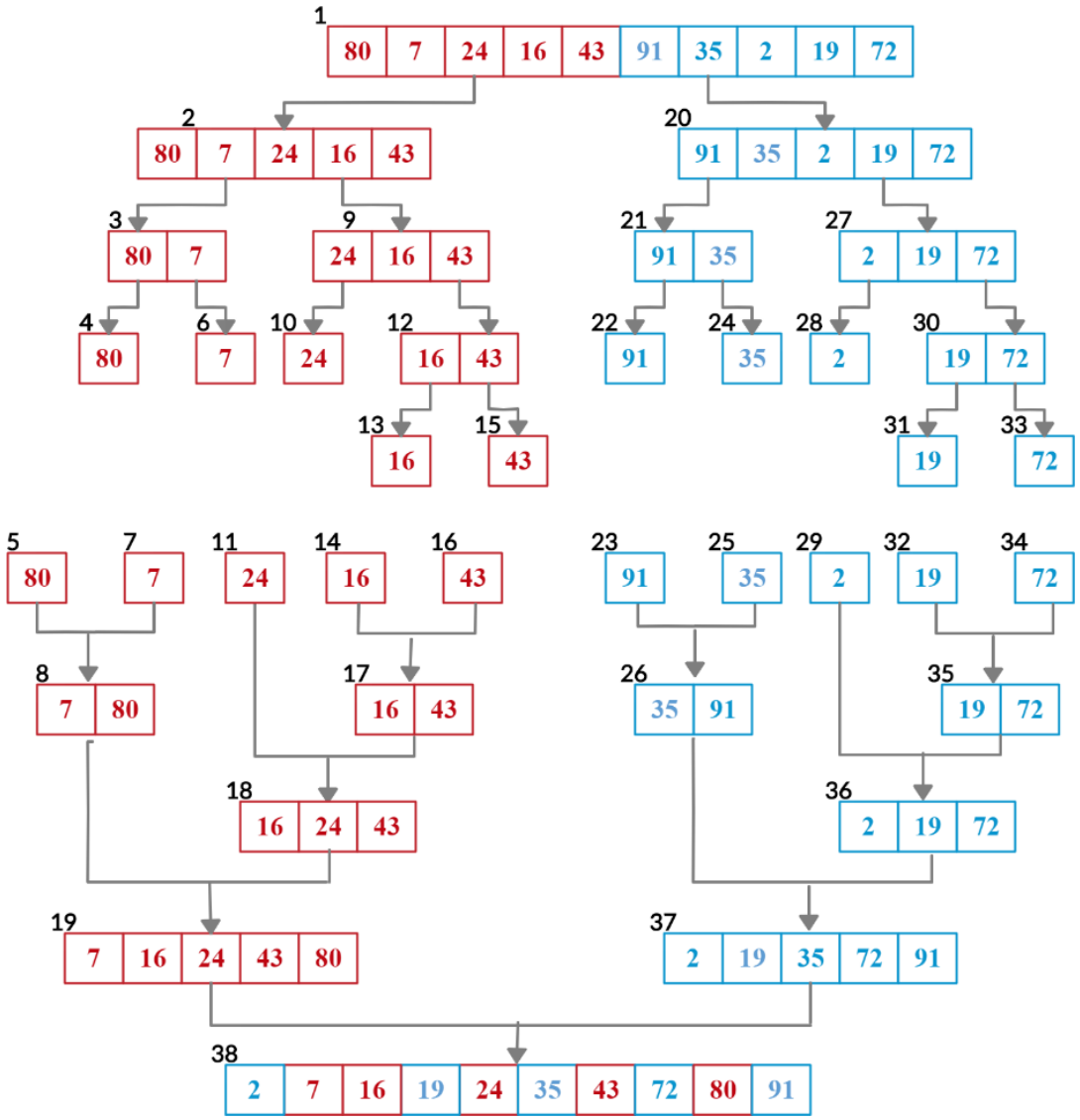
```
    def mergeSorted(self, list1, list2):
        if list1 is None:
            return list2
        if list2 is None:
            return list1

        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(15)
list1.appendSorted(22)
list1.appendSorted(4)
list1.appendSorted(12)
list1.appendSorted(8)
print("=================")
print("List 1 :"),
list1.printList()

list2 = LinkedList()
list2.appendSorted(8)
list2.appendSorted(11)
list2.appendSorted(2)
print("=================")
print("List 2 :"),
list2.printList()

list3 = LinkedList()
list3.head = list3.mergeSorted(list1.head, list2.head)
print("=================")
print("Merged List :"),
list3.printList()
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(data)
        else:
            ent = self.head
            currentent = self.head
            while currentent.next != None:
                currentent = currentent.next
            currentent.next = Node(data)
        return self.head

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

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

        if prev == None:
            self.head = node
        else:
            prev.next = node

        node.next = currentent

    def printList(self):
        currentent = self.head
        while currentent != None:
            print("%d" % currentent.data),
            currentent = currentent.next
```

Berikut output:

Nomor 2.
Beri nomer urut eksekusi proses gambar 6.1 dan 6.2 mengacu pada output di halaman 59



Nomor 4. A. diberikan List = [80,7,24,16,43,91,35,2,19,72] ,gambarlah trace pengurutan algoritmanya (Merge sort)



4 B. diberikan List = [80,7,24,16,43,91,35,2,19,72] ,gambarlah trace pengurutan algoritmanya (quickSort)

80	7	24	16	43	91	35	2	19	72
19	7	24	16	2	91	34	43	80	72
2	7	16	24	19	35	91	43	80	72
2	7	16	19	24	35	43	91	80	72
2	7	16	19	24	35	43	72	80	91