Nama : Windiapriani Ginayawati
NIM : L200170157
Kelas : D
Modul:VI

Nomor 1 Mengurutkan list yang berisi object-object class mhsTIF

```
( No. 1 )
class mhsTIF(object):
    def __init__(self, nama, NIM, kota, us):
        self.nama = nama
        self.NIM = NIM
        self.kotaTinggal = kota
        self.uangSaku = us

m0 = mhsTIF('Ana', 24, 'Sukoharjo', 240000)
m1 = mhsTIF('Ani', 45, 'Sragen', 230000)
m2 = mhsTIF('Aka', 32, 'Surakarta', 250000)
m3 = mhsTIF('Aki', 8, 'Surakarta', 235000)
m4 = mhsTIF('Silva',14, 'Sukoharjo', 240000)
m5 = mhsTIF('Silvi', 31, 'Salatiga', 250000)

Daftar = [m0, m1, m2, m3, m4, m5]

def urutnim(a):
    baru = {}
    for i in range(len(a)):
        baru[a[i].nama] = a[i].NIM
    listofTuples = sorted(baru.items(), key = lambda x: x[1])
    for elem in listofTuples:
        print (elem[0], ':', elem[1])
urutnim(Daftar)
```

Hasil Run

```
Aki : 8
Silva : 14
Ana : 24
Silvi : 31
Aka : 32
Ani : 45
```

Nomor 2

-

Nomor 3 Menguji kecepatan.

```
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]
        separuhKanan = A[mid:]
```

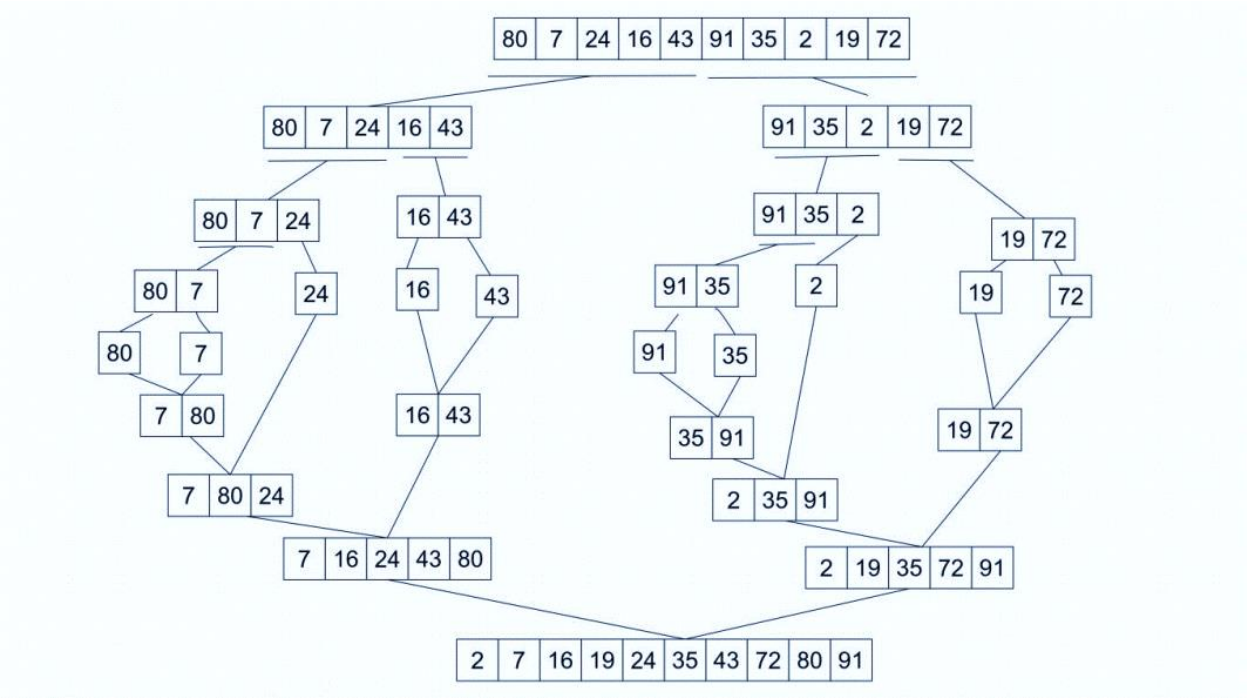
```
        mergeSort(separuhKiri)
        mergeSort(separuhKanan)

        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
```

```
    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 Sort : %g detik" %(ak-aw));
aw = detak();selectionSort(u_sel);ak=detak();print("Selection Sort : %g detik" %(ak-aw));
aw = detak();insertionSort(u_ins);ak=detak();print("Insertion Sort : %g detik" %(ak-aw));
aw = detak();mergeSort(u_mrg);ak=detak();print("Merge Sort: %g detik" %(ak-aw));
aw = detak();quickSort(u_qck);ak=detak();print("Quick Sort : %g detik" %(ak-aw));
```

Hasil Run

```
Bubble Sort : 3.91745 detik
Selection Sort : 3.68268 detik
Insertion Sort : 4.17221 detik
Merge Sort: 0.0676777 detik
Quick Sort : 0.0406075 detik
```

Nomor 4



Nomor 5 mergeSort secara rekursif

File Edit Format Run Options Window Help

```
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([5,2,7]))
```

Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 21:2
1)] on win32
Type "help", "copyright", "credits" or "license()"
>>>
= RESTART: D:/Semester 4/Praktikum Algoritma dan S
[2, 5, 7]
>>>
```

Nomor 6 Mengubah program quickSort dengan metode median-dari-tiga

```
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([2,24,5,52,70,31])

quickSort(listel, False)
print('sorted:')
print(listel)
```

Hasil Run

```
sorted:
[70, 52, 31, 24, 5, 2]
```

Nomor 7 Menguji kecepatan

```
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)
```

```
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));
```

Hasil run

```
merge : 0.062629 detik
quick : 0.0356338 detik
merge mod : -0.00347281 detik
quick mod : -0.106636 detik
```

Nomor 8 Linked List untuk mergeSort

```
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:
```

```
    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(11)
list1.appendSorted(2)
list1.appendSorted(31)
list1.appendSorted(8)
list1.appendSorted(84)

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

list2 = LinkedList()
list2.appendSorted(3)
list2.appendSorted(10)
list2.appendSorted(4)

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

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

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

Hasil run

```
List 1 :
2
8
11
31
84
List 2 :
3
4
10
Merged List :
2
3
4
8
10
11
31
84
```