ALGORITMA DAN STRUKTUR DATA
MODUL 6
PENGURUTAN LANJUTAN

Disusun oleh:
Muhammad Ferizal Fadhli
L200210119
D

TEKNIK INFORMATIKA
FAKULTAS KOMUNIKASI DAN INFORMATIKA
UNIVERSITAS MUHAMMADIYAH SURAKARTA
2022/2023

Soal-soal Mahasiswa

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

```
class MhsTIF(object):
    """Class MhsTIF yang dibangun dari class Mahasiswa"""

    def __init__(self,nama,NIM,kota,us):
        """Metode inisialisasi ini menutupimetode inisiasi di class
Manusia."""
        self.nama=nama
        self.NIM=NIM
        self.kotaTinggal=kota
        self.uangsaku=us


c0 = MhsTIF('Ika', 10, 'sukoharjo', 240000)
c1 = MhsTIF('Budi', 51, 'sragen', 230000)
c2 = MhsTIF('Ahmad', 2, 'surakarta', 250000)
c3 = MhsTIF('Chandra', 18, 'surakarta', 235000)
c4 = MhsTIF('Eka', 4, 'boyolali', 240000)
c5 = MhsTIF('Fandi', 31, 'salatiga', 250000)
c6 = MhsTIF('Deni', 13, 'klaten', 245000)
c7 = MhsTIF('Galuh', 5, 'wonogiri', 245000)
c8 = MhsTIF('Janto', 23, 'klaten', 245000)
c9 = MhsTIF('Hasan', 64, 'karanganyar', 270000)
c10 = MhsTIF('Khalid', 29, 'purwadadi', 265000)


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

def mergeSort(nlist):
    print("Membelah ",nlist)
    if len(nlist)>1:
        mid = len(nlist)//2
        lefthalf = nlist[:mid]
        righthalf = nlist[mid:]

        mergeSort(lefthalf)
        mergeSort(righthalf)
        i=j=k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i] < righthalf[j]:
                nlist[k]=lefthalf[i]
                i=i+1
```

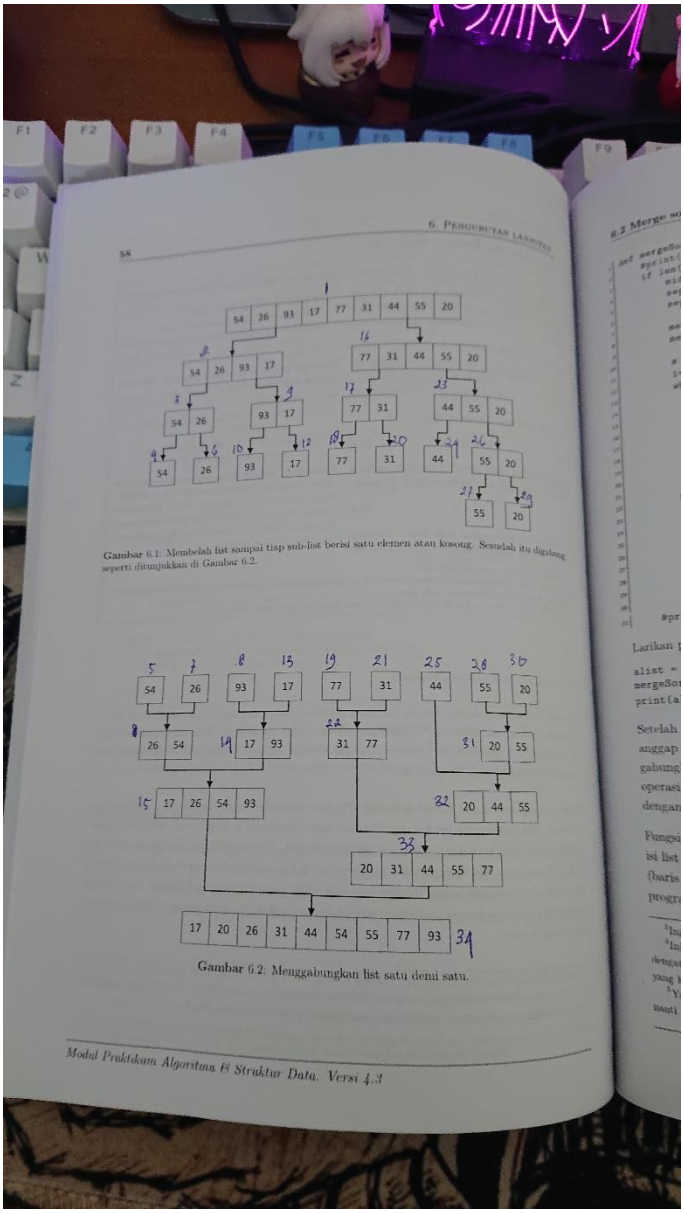
```
            else:
                nlist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            nlist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            nlist[k]=righthalf[j]
            j=j+1
            k=k+1
    print("Menggabungkan ",nlist)
nlist = Daftar
print("Hasil dari MergeSort")
mergeSort(nlist)
print(nlist)

def quickSort(data_list):
   quickSortHlp(data_list,0,len(data_list)-1)

def quickSortHlp(data_list,first,last):
   if first < last:

       splitpoint = partition(data_list,first,last)

       quickSortHlp(data_list,first,splitpoint-1)
       quickSortHlp(data_list,splitpoint+1,last)


def partition(data_list,first,last):
   pivotvalue = data_list[first]

   leftmark = first+1
   rightmark = last

   done = False
   while not done:

       while leftmark <= rightmark and data_list[leftmark] <=
pivotvalue:
           leftmark = leftmark + 1

       while data_list[rightmark] >= pivotvalue and rightmark >=
leftmark:
           rightmark = rightmark -1
```

```
        if rightmark < leftmark:
            done = True
        else:
            temp = data_list[leftmark]
            data_list[leftmark] = data_list[rightmark]
            data_list[rightmark] = temp

    temp = data_list[first]
    data_list[first] = data_list[rightmark]
    data_list[rightmark] = temp


    return rightmark

data_list = Daftar
quickSort(data_list)
print("\n"+"Hasil QuickSort")
print(data_list)
```

Berikut SS kode program setelah dijalankan :

```
======Hasil Nomor 1=======
Hasil dari MergeSort
Membelah  [10, 51, 2, 18, 4, 31, 13, 5, 23, 64, 29]
Membelah  [10, 51, 2, 18, 4]
Membelah  [10, 51]
Membelah  [10]
Menggabungkan  [10]
Membelah  [51]
Menggabungkan  [51]
Menggabungkan  [10, 51]
Membelah  [2, 18, 4]
Membelah  [2]
Menggabungkan  [2]
Membelah  [18, 4]
Membelah  [18]
Menggabungkan  [18]
Membelah  [4]
Menggabungkan  [4]
Menggabungkan  [4, 18]
[2, 4, 5, 10, 13, 18, 23, 29, 31, 51, 64]

Hasil QuickSort
[2, 4, 5, 10, 13, 18, 23, 29, 31, 51, 64]
```

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.



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.

Modul Praktikum Algoritma & Struktur Data, Versi 4.3

3. Uji kecepatan. Ujilah mergeSort dan quickSort di atas (bersama metode sort yang kamu pelajari sebelumnya) dengan kode di bawah ini.
   Berikut kode program ;

```
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 bubb(arr):
    n = len(arr)
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1] :
                arr[j], arr[j+1] = arr[j+1], arr[j]
```

```
def sele(A):
    for i in range(len(A)):
        min_idx = i
        for j in range(i+1, len(A)):
            if A[min_idx] > A[j]:
                min_idx = j
        A[i], A[min_idx] = A[min_idx], A[i]

def inse(arr):
    for i in range(1, len(arr)):
        key = arr[i]
        j = i-1
        while j >=0 and key < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = key
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)


bub = k[:]
sel = k[:]
ins = k[:]
mer = k[:]
qui = k[:]

aw=detak();bubb(bub);ak=detak();print('bubble : %g detik' %(ak-aw));
aw=detak();sele(sel);ak=detak();print('selection : %g detik' %(ak-aw));
aw=detak();inse(ins);ak=detak();print('insertion : %g detik' %(ak-aw));
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));
```

Berikut SS Ketika program dijalankan :

```
bubble : 8.39712 detik
selection : 4.16875 detik
insertion : 4.6139 detik
merge : 0.0749063 detik
quick : 0.0443275 detik
```

4. Diberikan list L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72], gambarlah trace pengurutan10 untuk algoritma.
   a. Marge sort

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						72
				2	35	43	91		19

Proses 3

2	7	16	24	35	43	80	91		72	19

Proses 4

2	7	16	19	24	35	43	72	80	91

b. Quick sort

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

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 quick sort.
   Berikut kode 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 merge_sort(the_list):
    return _merge_sort((0, len(the_list) - 1), the_list)

print(merge_sort([13,45,12,3,10,2]))
```

Berikut SS program ketikaa dijalankana :

```
====Hasil Nomor 5=====
[2, 3, 10, 12, 13, 45]
```

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

```
print("\n"+"====Hasil 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([14,4,2,104,23,50])

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

Berikut SS program Ketika dijalankan :

```
sorted:
[104, 50, 23, 14, 4, 2]
```

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

```
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));
```

Berikut SS program Ketika dijalankann :

```
merge : 0.104664 detik
quick : 0.0512075 detik
merge mod : 0 detik
quick mod : -0.109097 detik
3
7
9
10
12
13
16
```

8. Buatlah versi linked-list untuk program mergeSort di atas.
   Berikut kode program :

```
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()
```

Berikut SS Ketika program dijalankan :