

Nama : Alfianto Andy P

NIM : L200180194

## MODUL 6

Modul 6.py - D:\Materi smester 4\Algo struktur data\Modul 6.py (3.6.5)

File Edit Format Run Options Window Help

```
#NO 1
print ('No 1: ')
class MhsTIF(object) :
    def __init__(self, nama, nim, asal, uangsaku) :
        self.nama = nama
        self.nim = nim
        self.asal = asal
        self.uangsaku = uangsaku

m0 = MhsTIF('Alfianto', 9, 'Boylali', 300000)
m1 = MhsTIF('Hari', 10, 'Semarang', 320000)
m2 = MhsTIF('Mifta', 23, 'Kartasura', 350000)
m3 = MhsTIF('Desi', 45, 'Solo', 290000)
m4 = MhsTIF('Dewi', 27, 'Karanganyar', 310000)
m5 = MhsTIF('Lia', 56, 'Wonogiri', 380000)
m6 = MhsTIF('Bagus', 2, 'Boyolali', 280000)
m7 = MhsTIF('Wahyu', 8, 'Sragen', 330000)
m8 = MhsTIF('Lusiana', 34, 'Purwodadi', 340000)
m9 = MhsTIF('Alfina', 60, 'Sleman', 390000)
m10 = MhsTIF('Akbar', 51, 'Magelang', 370000)

urut =[m0.nim, m1.nim, m2.nim, m3.nim, m4.nim, m5.nim,
        m6.nim, m7.nim, m8.nim, m9.nim, m10.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):
```

```

        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 = urut
print("Hasil 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

```

Modul 6.py - D:\Materi smester 4\Algo struktur data\Modul 6.py (3.6.5)

File Edit Format Run Options Window Help

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

#NO 3
print('No 3:')
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 bubbleSort(X) :
    n = len(X)
    for i in range(n):
        for j in range(0, n-i-1):
            if X[j] > X[j+1] :
                X[j], X[j+1] = X[j+1], X[j]

def selectionSort(X) :
    for i in range(len(X)):
        min_idk = i
        for j in range(i+1, len(X)):
            if X[min_idk] > X[j]:
                min_idk = j
        X[i], X[min_idk] = X[min_idk], X[i]

def insertSort(X) :
    n = len (X)
    for i in range (1, n) :
```

Modul 6.py - D:\Materi smester 4\Algo struktur data\Modul 6.py (3.6.5)

File Edit Format Run Options Window Help

```
def mergeSort(X):
    if len(X) > 1:
        mid = len(X) // 2
        L = X[:mid]
        R = X[mid:]
        mergeSort(L)
        mergeSort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                X[k] = L[i]
                i += 1
            else:
                X[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            X[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            X[k] = R[j]
            j += 1
            k += 1
def partition(X, low, high):
    i = ( low-1 )
    pivot = X[high]
    for j in range(low , high):
        if X[j] <= pivot:
            i = i+1
            X[i],X[j] = X[j],X[i]
    X[i+1],X[high] = X[high],X[i+1]
    return ( i+1 )
```

```
def mergeSort(X, low, high):
```

```

def quickSort(X, low, high):
    if low < high:
        pi = partition(X, low, high)
        quickSort(X, low, pi-1)
        quickSort(X, pi+1, high)

u_bub = k[:]
u_sel = k[:]
u_ins = k[:]
u_mer = 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 () ; insertSort (u_ins) ; ak = detak() ; print('insert : % g detik' % (ak - aw)) ;
aw = detak () ; mergeSort (u_mer) ; ak = detak() ; print('merge : % g detik' % (ak - aw)) ;
aw = detak () ; quickSort (u_qck, 0, len(u_qck)-1) ; ak = detak() ; print('quick : % g detik' % (ak - aw)) ;

# NO 5
print ('No 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]

```

```

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]))

# NO 6
print ('No 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
    j = high
    while i <= j:
        while i <= j and L[i] <= pivot:
            i += 1
        while i <= j and L[j] > pivot:
            j -= 1
        if i < j:
            L[i], L[j] = L[j], L[i]
    L[low], L[j] = L[j], L[low]
    result += 1
    return j, result

```

```

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([14,4,2,104,23,50])

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

# NO 7
print ('No 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

```

```
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]
```



Modul 6.py - D:\Materi smester 4\Algo struktur data\Modul 6.py (3.6.5)

File Edit Format Run Options Window Help

```
all[i+1], all[high] = all[high], all[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
    return new_list
```

```

        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

```

```

11 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));

# NO 8
print ('Nom 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

```

Modul 6.py - D:\Materi smester 4\Algo struktur data\Modul 6.py (3.6.5)

File Edit Format Run Options Window Help

```
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()
```

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

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

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

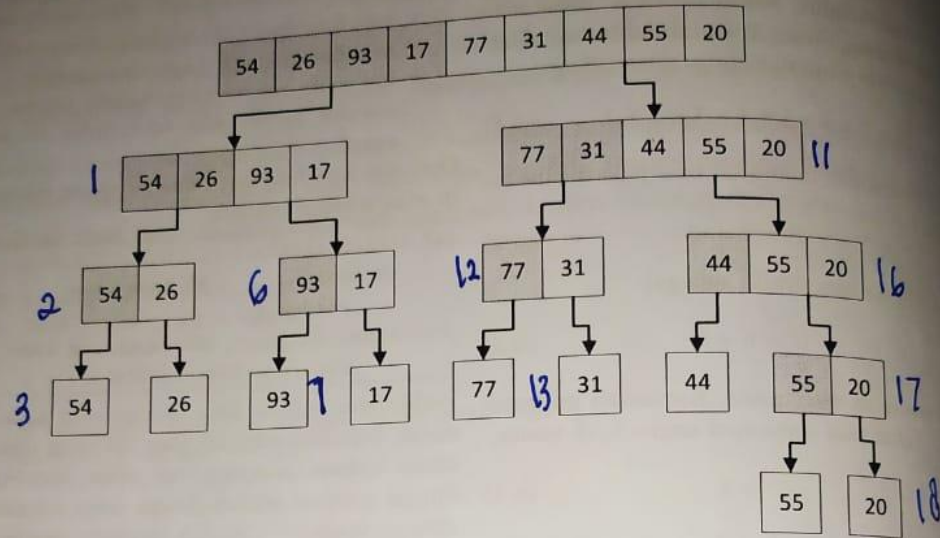
list3 = LinkedList()
list3.head = list3.mergeSorted(list1.head, list2.head)

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

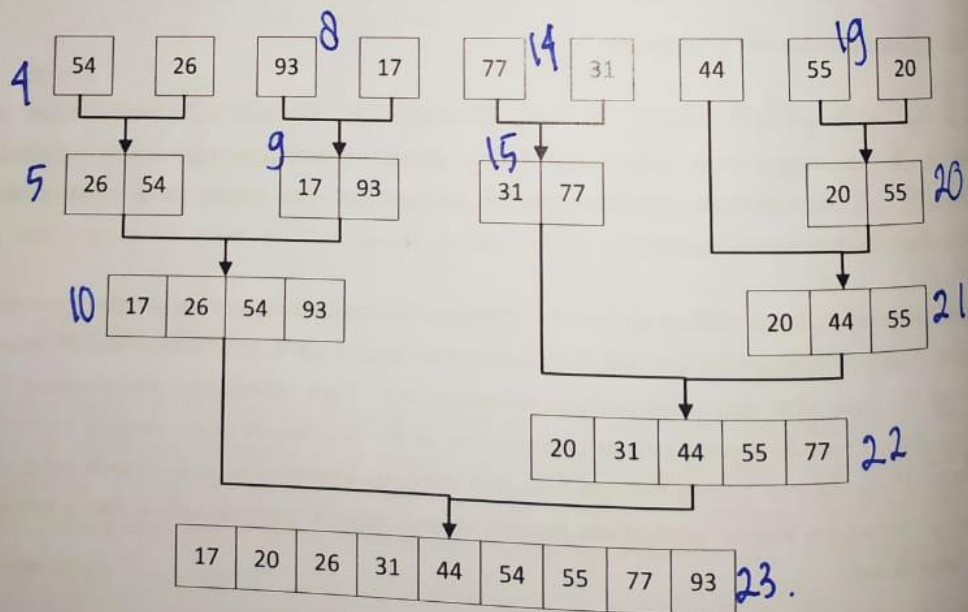
---

No 2

58



**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.

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
No 1:
Hasil MergeSort
Membelah [9, 10, 23, 45, 27, 56, 2, 8, 34, 60, 51]
Membelah [9, 10, 23, 45, 27]
Membelah [9, 10]
Membelah [9]
Menggabungkan [9]
Membelah [10]
Menggabungkan [10]
Menggabungkan [9, 10]
Membelah [23, 45, 27]
Membelah [23]
Menggabungkan [23]
Membelah [45, 27]
Membelah [45]
Menggabungkan [45]
Membelah [27]
Menggabungkan [27]
Menggabungkan [27, 45]
Menggabungkan [23, 27, 45]
Menggabungkan [9, 10, 23, 27, 45]
Membelah [56, 2, 8, 34, 60, 51]
Membelah [56, 2, 8]
Membelah [56]
Menggabungkan [56]
Membelah [2, 8]
Membelah [2]
Menggabungkan [2]
Membelah [8]
Menggabungkan [8]
Menggabungkan [2, 8]
Menggabungkan [2, 8, 56]
Membelah [34, 60, 51]
Membelah [34]
Menggabungkan [34]
Membelah [60, 51]
Membelah [60]
Menggabungkan [60]
Membelah [51]
Menggabungkan [51]
Menggabungkan [51, 60]
Menggabungkan [34, 51, 60]
Menggabungkan [2, 8, 34, 51, 56, 60]
Menggabungkan [2, 8, 9, 10, 23, 27, 34, 45, 51, 56, 60]
```

Python 3.6.5 Shell

File Edit Shell Debug Options Window Help

```
Menggabungkan [34, 51, 60]
Menggabungkan [2, 8, 34, 51, 56, 60]
Menggabungkan [2, 8, 9, 10, 23, 27, 34, 45, 51, 56, 60]
[2, 8, 9, 10, 23, 27, 34, 45, 51, 56, 60]
```

Hasil QuickSort

```
[2, 8, 9, 10, 23, 27, 34, 45, 51, 56, 60]
```

No 3:

```
bubble : 7.84153 detik
selection : 2.98223 detik
insert : 0.031275 detik
merge : 0.0781293 detik
quick : 0.0312376 detik
```

No 5

```
[2, 3, 10, 12, 13, 45]
```

No 6

sorted:

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

No 7

```
merge : 0.110115 detik
quick : 0.0402908 detik
merge mod : -0.00609565 detik
quick mod : -0.120969 detik
```

Nom 8

List 1 :

```
3
7
12
13
16
```

List 2 :

```
1
9
10
```

Merged List :

```
1
3
7
9
10
12
13
16
```

```
>>> |
```



