NAMA : AGATHA FEBIANANDA P

NIM : L200170127

KELAS : D

MODUL : 6

1

```
#Nomer 1
from t2 import mahasiswa
from kode import urut

m1=mahasiswa("Aiza", 144, "Samarinda", 250000)
m2=mahasiswa("Bella", 158, "Jakarta", 350000)
m3=mahasiswa("Chiara", 124, "Bontang", 220000)
m4=mahasiswa("Deena", 104, "Cimahi", 200000)
m5=mahasiswa("Elvira", 120, "Magetan", 205000)

nimMH = [m1.nim, m2.nim, m3.nim, m4.nim, m5.nim]
usMH = [m1.us, m2.us, m3.us, m4.us, m5.us]

a1 = urut(nimMH)
b2 = urut(usMH)

a1.printMerge(nimMH)
b2.printMerge(usMH)

a1.printQuick(nimMH)
b2.printQuick(usMH)
```

```
Merge sort
104 120 124 144 158

Merge sort
200000 205000 220000 250000 350000

Quick sort
104 120 124 144 158

Quick sort
200000 205000 220000 250000 350000
```

2 -

3.

```python
#Nomer 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 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
```

```python
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)
```

```python
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));
```

```
bubble : 5.92284 detik
selection : 2.03281 detik
insertion : 2.55388 detik
merge : 0.0520144 detik
quick : 0.0357845 detik
```

4.

a. Merge sort

b. Quick sort

    -

5.
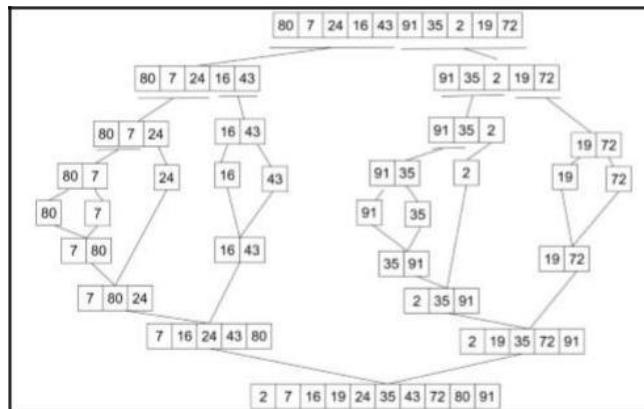
```python
#Nomer 5
import random
def _merge_sort(indices, the_list):
    start = indices[0]
    end = indices[1]
    half_way = (end - start)//2 + start
    if start < half_way:
        _merge_sort((start, half_way), the_list)
    if half_way + 1 <= end and end - start != 1:
        _merge_sort((half_way + 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
```

```python
def merge_sort(the_list):
    return _merge_sort((0, len(the_list) - 1), the_list)

print(merge_sort([28,3,1, 9, 99]))
```

```
[1, 3, 9, 28, 99]
```

6.

```python
#Nomer 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
```

```python
listt = list([1,34,65,24,53])

quickSort(listt, False)  # des
print('sorted:')
print(listt)
```

```
sorted:
[65, 53, 34, 24, 1]
```

7.

```python
#Nomer 7
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
```

```python
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])
```

```python
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
```

```python
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));
```

```
merge : 0.0480859 detik
quick : 0.0298214 detik
merge mod : -0.00905609 detik
quick mod : -0.0645561 detik
```

8.

```python
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
```

```python
  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()
```

```
List 1 :
3
7
12
13
16
List 2 :
1
9
10
Merged List :
1
3
7
9
10
12
13
16
```