**PRAKTIKUM ALGORITMA STRUKTUR DATA**

**MODUL 6**

**PENGATURAN LANJUTAN**

****

**Disusun oleh:**

**Adinda Aulia Hapsari**

**L200220037**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**FAKULTAS KOMUNIKASI DAN INFORMATIKA**

**UNIVERSITAS MUHAMMADIYAH SURAKARTA**

**TAHUN 2024**

**6.4 Soal-soal untuk Mahasiswa**

Sebelum mengerjakan soal-soal dibawah, kerjakan dulu latihan-latihan diatas.

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

print('\n--- Oleh L200220037 ---')

print ( "Nomor 1")

class Mahasiswa(object):

    def \_\_init\_\_ (self,nim) :

        self.nim = nim

a1= "L200170156"

a2= "L200170152"

a3= "L200170155"

a4= "L200170147"

a5= "L200170143"

Daftar = [a1,a2,a3,a4,a5]

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

mergeSort(Daftar)

print("Menggunakan Merge Sort : \n",Daftar)

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

        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]

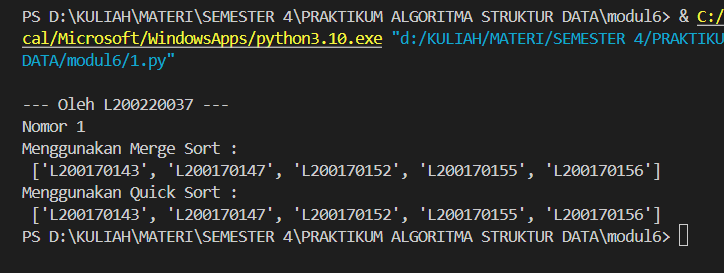
    A[penandaKanan] = temp

    return penandaKanan

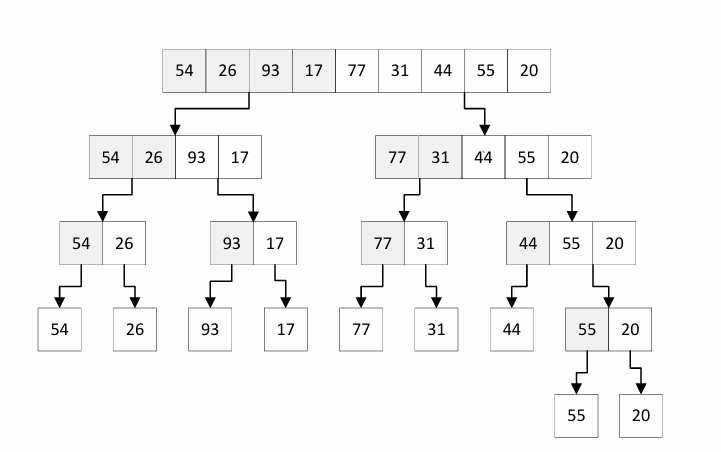
quickSort(Daftar)

print("Menggunakan Quick Sort : \n",Daftar)

Output:



1. Memakai bolpen merah atau biru, tandai dan beri nomer urut eksekusi proses pada Gam bar 6.1 dan 6.2, dengan mengacu pada output dihalaman 59.



**17**

**16**

**15**

**14**

**13**

**12**

**11**

**10**

* **1**

**9**

**8**

**7**

**6**

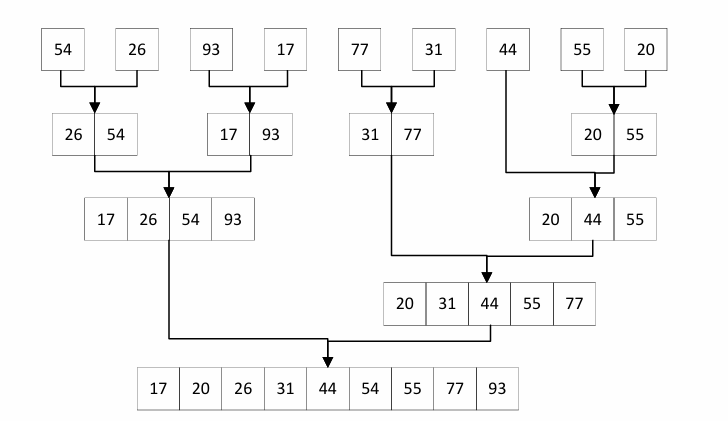
**5**

**4**

**3**

**2**

**1**



**5**

**3**

**4**

**3**

**2**

**1**

1. Uji kecepatan. Ujilah mergeSort dan quickSort diatas (bersama metode sort yang kamu pelajari sebelumnya) dengan kode dibawah ini.

from time import time as detak

from random import shuffle as kocok

import time

k = range(6000)

kocok(k)

u\_bub = k[:] ##

u\_sel = k[:] ## Deep copy.

u\_ins = k[:] ## Jangan lupa [:]-nya!

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-aw) );

aw=detak();mergeSort(u\_mrg);ak=detak();print(’merge: %g detik’ %(ak-aw) );

aw=detak();quickSort(u\_qck);ak=detak();print(’quick: %g detik’ %(ak-aw) );

Tunjukkan hasil ujinya keasisten praktikum.

print('\n--- Oleh L200220037 ---')

print ("Nomor 3")

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

        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 : %g detik" %(ak-aw));

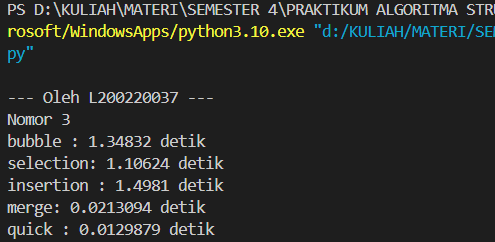
aw = detak();selectionSort(u\_sel);ak=detak();print("selection: %g detik" %(ak-aw));

aw = detak();insertionSort(u\_ins);ak=detak();print("insertion : %g detik" %(ak-aw));

aw = detak();mergeSort(u\_mrg);ak=detak();print("merge: %g detik" %(ak-aw));

aw = detak();quickSort(u\_qck);ak=detak();print("quick : %g detik" %(ak-aw));

Output:



1. Diberikan list L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72], gambarlah trace pengurutan untuk algoritma
2. Mergesort

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |  | 2 | 35 | 43 | 91 |  | 19 | 72 |

Proses 3

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 7 | 16 | 24 | 35 | 43 | 80 | 91 |  | 19 | 72 |

Proses 4

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 | 7 | 16 | 19 | 24 | 35 | 43 | 72 | 80 | 91 |

1. Quicksort

pivot

high pivot

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 80 | 7 | 24 | 16 | 43 | 91 | 35 | 2 | 19 | 72 |

low

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 7 | 24 | 16 | 43 | 91 | 35 | 2 | 19 | 80  high pivot |

low

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 7 | 24 | 16 | 43 | 91  low pivot | 35 | 2 | 19 | 80  high |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 7 | 24 | 16 | 43 | 80  low | 35 | 2 | 19 | 91  high |

pivot

high

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 72 | 7 | 24 | 16 | 43 | 19  low | 35 | 2 | 80 | 91 |

**Soal-soal dibawah ini sedikit lebih sulit. Kerjakanlah dirumah.**

1. 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 quickSort.

print('\n--- Oleh L200220037 ---')

print ("Nomor 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

    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

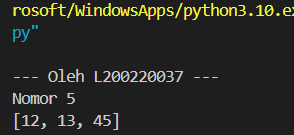
    return the\_list

def merge\_sort(the\_list):

    return \_merge\_sort((0, len(the\_list) - 1), the\_list)

print(merge\_sort([13,45,12]))

Output:



1. Apakah kita bisa meningkatkan efisiensi program quicksort dengan memakai metode median-dari-tiga untuk memilih pivotnya? Ubahlah kodenya dan ujilah.

print('\n--- Oleh L200220037 ---')

print("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

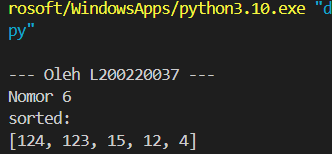
liste1 = list([12,4,15,124,123])

quickSort(liste1, False)  # descending order

print('sorted:')

print(liste1)

Output:



1. Uji-kecepatan keduanya dan perbandingkan juga dengan kode awalnya.

print('\n--- Oleh L200220037 ---')

print("Nomor 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

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[:]

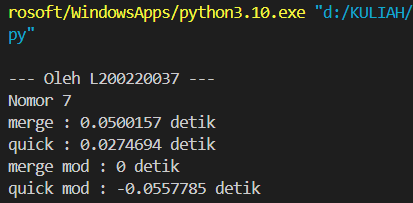
aw=detak();mergeSort(mer);ak=detak();print('merge : %g detik' %(ak-aw));

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));

Output:



1. Buatlah versi linked-list untuk program mergeSort diatas.

print('\n--- Oleh L200220037 ---')

print("Nomor 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

    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()

Output:

