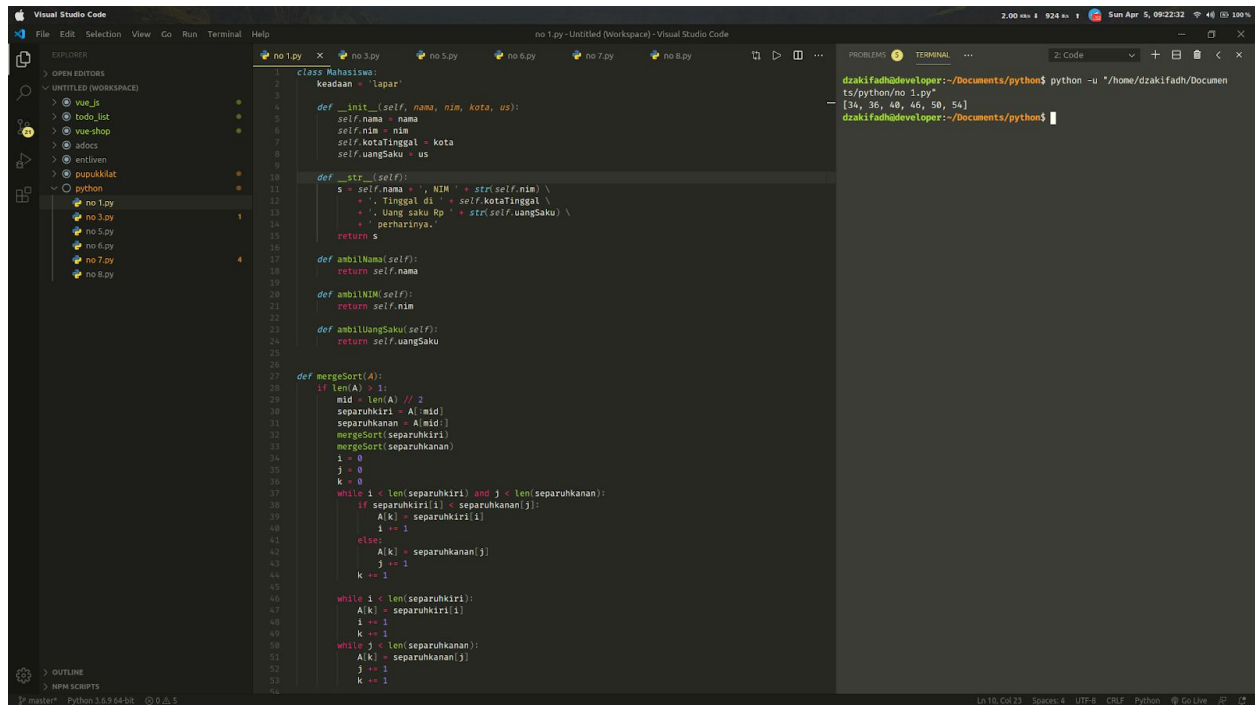


Nama : Dzaki Fadhlurrohan
NIM : L200180064
Kelas : C

Modul 6

Pengurutan Lanjutan

Nomor 1



The screenshot shows the Visual Studio Code interface with a Python file named 'no 1.py'. The code defines a 'Mahasiswa' class and a 'mergeSort' function. The class has attributes 'nama', 'nim', 'kota', and 'uangSaku', and methods to retrieve these values. The 'mergeSort' function implements a recursive merge sort algorithm.

```
class Mahasiswa:
    keadaan = 'lapar'

    def __init__(self, nama, nim, kota, us):
        self.nama = nama
        self.nim = nim
        self.kotaTinggal = kota
        self.uangSaku = us

    def __str__(self):
        s = self.nama + ', NIM ' + str(self.nim) \
            + '. Tinggal di ' + self.kotaTinggal \
            + '. Uang saku Rp ' + str(self.uangSaku) \
            + ' perharinya.'
        return s

    def ambilNama(self):
        return self.nama

    def ambilNIM(self):
        return self.nim

    def ambilUangSaku(self):
        return self.uangSaku

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 += 1
            else:
                A[k] = separuhkanan[j]
                j += 1
            k += 1
        while i < len(separuhkiri):
            A[k] = separuhkiri[i]
            i += 1
            k += 1
        while j < len(separuhkanan):
            A[k] = separuhkanan[j]
            j += 1
            k += 1
```

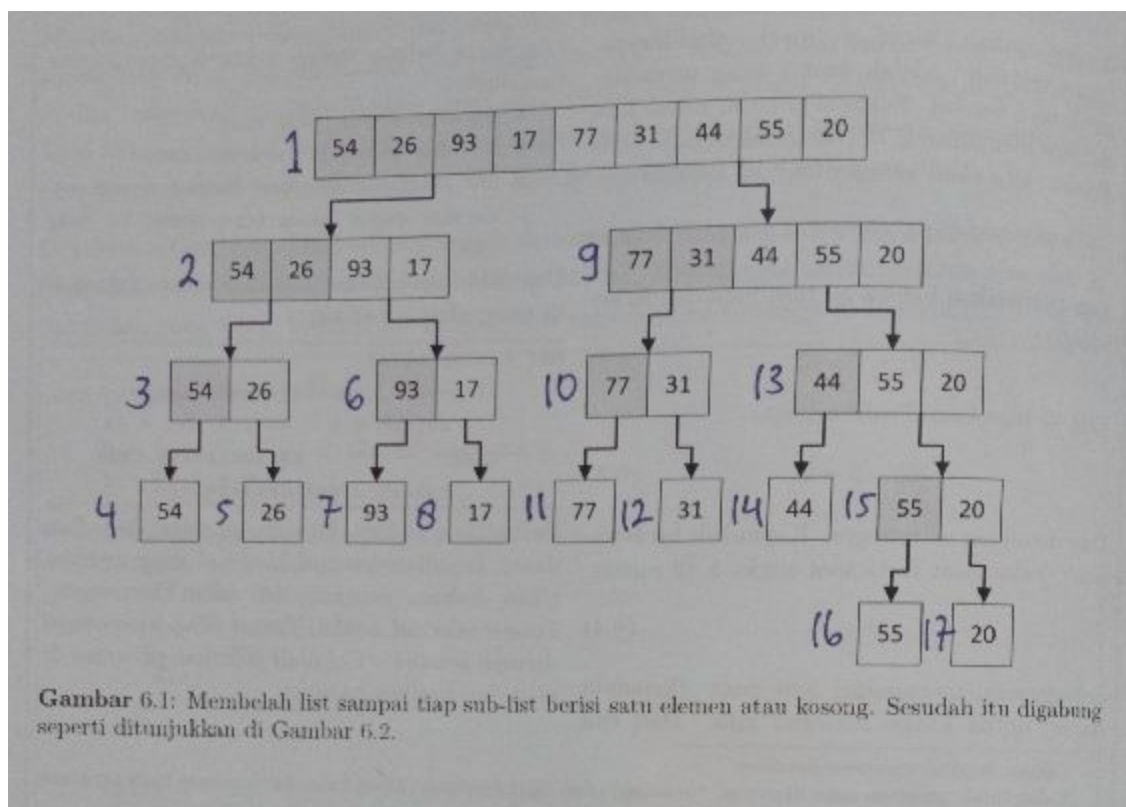
```
54
55
56 def quickSort(A):
57     quickSortBantu(A, 0, len(A) - 1)
58
59
60 def quickSortBantu(A, awal, akhir):
61     if awal < akhir:
62         titikBelah = partisi(A, awal, akhir)
63         quickSortBantu(A, awal, titikBelah - 1)
64         quickSortBantu(A, titikBelah + 1, akhir)
65
66
67 def partisi(A, awal, akhir):
68     nilaiPivot = A[awal]
69     penandaKiri = awal + 1
70     penandaKanan = akhir
71     selesai = False
72     while not selesai:
73         while penandaKiri < penandaKanan and A[penandaKiri] < nilaiPivot:
74             penandaKiri = penandaKiri + 1
75         while A[penandaKanan] > nilaiPivot and penandaKanan > penandaKiri:
76             penandaKanan = penandaKanan - 1
77         if penandaKanan < penandaKiri:
78             selesai = True
79         else:
80             temp = A[penandaKiri]
81             A[penandaKiri] = A[penandaKanan]
82             A[penandaKanan] = temp
83             temp = A[awal]
84             A[awal] = A[penandaKiri]
85             A[penandaKanan] = temp
86             return penandaKanan
87
88
89 mahasi = Mahasiswa("Doaki", 28, "Jakarta", 255000)
90 mahas2 = Mahasiswa("Jhon", 36, "NY", 600000)
91 mahas3 = Mahasiswa("Doe", 40, "California", 450000)
92 mahasi = Mahasiswa("Mike", 46, "Washington DC", 200000)
93 mahas5 = Mahasiswa("Jhon", 50, "NY", 350000)
94 mahas6 = Mahasiswa("Brad", 54, "Texas", 470000)
95
96 listin = [mahasi.nim, mahas2.nim, mahas3.nim,
97           mahas4.nim, mahas5.nim, mahas6.nim]
98 mergeSort(listin)
99 print(listin)
100
```

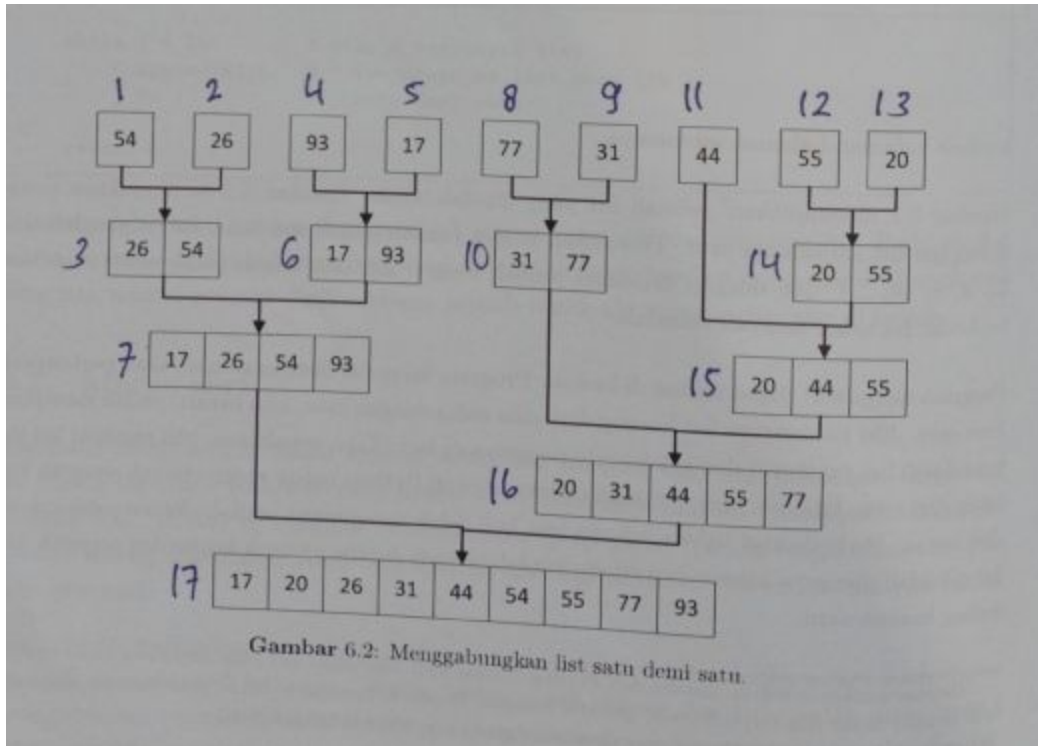
dzakifadh@developer:~/Documents/python\$ python -u "/home/dzakifadh/Documents/python/no 1.py"

[34, 36, 48, 46, 50, 54]

dzakifadh@developer:~/Documents/python\$

Nomor 2





Nomor 3

```

Visual Studio Code
no 3.py -Untitled (Workspace) -Visual Studio Code
668 x 468 100% Sun Apr 5, 09:25:46

EXPLORER
  OPEN EDITORS
  UNTITLED (WORKSPACE)
  @ vue.js
  @ todo_list
  @ vue-shop
  @ select
  @ editview
  @ pupaklat
  python
    no 1.py
    no 3.py
    no 5.py
    no 6.py
    no 7.py
    no 8.py

1 from time import time as detik
2 from random import shuffle as kocok
3
4
5 def swap(a, p, q):
6     temp = a[p]
7     a[p] = a[q]
8     a[q] = temp
9
10
11 def cariposisiterkecil(A, dariini, sampainini):
12     posisiterkecil = dariini
13     for i in range(dariini + 1, sampainini):
14         if A[i] < A[posisiterkecil]:
15             posisiterkecil = i
16     return posisiterkecil
17
18
19 def bubbleSort(A):
20     n = len(A)
21     for i in range(n - 1):
22         for j in range(n - 1 - i):
23             if A[j] > A[j + 1]:
24                 swap(A, j, j + 1)
25
26
27 def selectionSort(A):
28     n = len(A)
29     for i in range(n - 1):
30         indexkecil = cariposisiterkecil(A, i, n)
31         if indexkecil != i:
32             swap(A, i, indexkecil)
33
34
35 def insertionSort(A):
36     n = len(A)
37     for i in range(1, n):
38         nilai = A[i]
39         pos = i
40         while pos > 0 and nilai < A[pos - 1]:
41             A[pos] = A[pos - 1]
42             pos = pos - 1
43         A[pos] = nilai
44
45
46 def mergeSort(A):
47     if len(A) > 1:
48         mid = len(A) // 2
49         L = A[:mid]
50         R = A[mid:]
51         mergeSort(L)
52         mergeSort(R)
53         i = j = k = 0
54         while i < len(L) and j < len(R):
55
PROBLEMS
TERMINAL
2: Code
dzakifadh@developer:~/Documents/python$ python -u ~/home/dzakifadh/Document
ts/python/no 3.py*
bubble : 2.25352 detik
selection : 0.713781 detik
insertion : 0.832365 detik
merge : 0.0194371 detik
quick : 0.00930485 detik
dzakifadh@developer:~/Documents/python$

```

```
def mergeSort(A):
    if len(A) > 1:
        mid = len(A) // 2
        L = A[:mid]
        R = A[mid:]
        mergeSort(L)
        mergeSort(R)
        i = j = k = 0
        while i < len(L) and j < len(R):
            if L[i] < R[j]:
                A[k] = L[i]
                i += 1
            else:
                A[k] = R[j]
                j += 1
            k += 1
        while i < len(L):
            A[k] = L[i]
            i += 1
            k += 1
        while j < len(R):
            A[k] = R[j]
            j += 1
            k += 1

def partition(A, low, high):
    i = (low + 1)
    pivot = A[high]
    for j in range(low, high):
        if A[j] <= pivot:
            i = i + 1
            A[i], A[j] = A[j], A[i]
    A[i + 1], A[high] = A[high], A[i + 1]
    return i + 1

def quickSortBantu(A, low, high):
    if low < high:
        pi = partition(A, low, high)
        quickSortBantu(A, low, pi - 1)
        quickSortBantu(A, pi + 1, high)

def quickSort(A):
    quickSortBantu(A, 0, len(A)-1)

k = [i for i in range(1, 6000)]
kocok(k)
bub = k[:]
sel = k[:]
ins = k[:]
```

```
dzakifadh@developer:~/Documents/python$ python -u "/home/dzakifadh/Document
s/python/no 3.py"
bubble : 2.25352 detik
selection : 0.713781 detik
insertion : 0.832365 detik
merge : 0.0194371 detik
quick : 0.00930405 detik
dzakifadh@developer:~/Documents/python$
```

```
def partition(A, low, high):
    i = (low + 1)
    pivot = A[high]
    for j in range(low, high):
        if A[j] <= pivot:
            i = i + 1
            A[i], A[j] = A[j], A[i]
    A[i + 1], A[high] = A[high], A[i + 1]
    return i + 1

def quickSortBantu(A, low, high):
    if low < high:
        pi = partition(A, low, high)
        quickSortBantu(A, low, pi - 1)
        quickSortBantu(A, pi + 1, high)

def quickSort(A):
    quickSortBantu(A, 0, len(A)-1)

k = [i for i in range(1, 6000)]
kocok(k)
bub = k[:]
sel = k[:]
ins = k[:]
mer = k[:]
qui = k[:]

aw = detak()
bubbleSort(bub)
ak = detak()
print("bubble : %g detik" % (ak-aw))
aw = detak()
selectionSort(sel)
ak = detak()
print("selection : %g detik" % (ak-aw))
aw = detak()
insertionSort(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)
ak = detak()
print("quick : %g detik" % (ak-aw))
```

```
dzakifadh@developer:~/Documents/python$ python -u "/home/dzakifadh/Document
s/python/no 3.py"
bubble : 2.25352 detik
selection : 0.713781 detik
insertion : 0.832365 detik
merge : 0.0194371 detik
quick : 0.00930405 detik
dzakifadh@developer:~/Documents/python$
```

Nomor 4

4(a). Merge Sort

list $L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]$

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
---	---	----	----	----	----	----	----	----	----

4(b). Quick Sort

List $L = [80, 7, 24, 16, 43, 91, 35, 2, 19, 72]$

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

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

high

Pivot

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

low

high

Pivot

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

low

high


```
1 def _merge_sort(indices, the_list):
2     start = indices[0]
3     end = indices[1]
4     half_way = (end - start) // 2 + start
5     if start < half_way:
6         _merge_sort((start, half_way), the_list)
7     if half_way + 1 < end and end - start >= 1:
8         _merge_sort((half_way + 1, end), the_list)
9     sort_sub_list(the_list, indices[0], indices[1])
10    return the_list
11
12 def sort_sub_list(the_list, start, end):
13    orig_start = start
14    initial_start_second_list = (end - start) // 2 + start + 1
15    list2_first_index = initial_start_second_list
16    new_list = []
17    while start < initial_start_second_list and list2_first_index <= end:
18        first1 = the_list[start]
19        first2 = the_list[list2_first_index]
20        if first1 < first2:
21            new_list.append(first1)
22            list2_first_index += 1
23        else:
24            new_list.append(first1)
25            start += 1
26    while start < initial_start_second_list:
27        new_list.append(the_list[start])
28        start += 1
29    while list2_first_index <= end:
30        new_list.append(the_list[list2_first_index])
31        list2_first_index += 1
32    for i in new_list:
33        the_list[orig_start] = i
34        orig_start += 1
35    return the_list
36
37 def merge_sort(the_list):
38     return _merge_sort((0, len(the_list) - 1), the_list)
39
40 print(merge_sort([15, 5, 7, 6, 80, 32, 4, 18, 76, 180]))
41
```

Nomor 6

```
1 def quickSort(S):
2     quicksorthelp(S, 0, len(S))
3
4 def quicksorthelp(S, low, high):
5     result = 0
6     if low < high:
7         pivot_location, result = Partition(S, low, high)
8         result += quicksorthelp(S, low, pivot_location)
9         result += quicksorthelp(S, pivot_location + 1, high)
10    return result
11
12 def Partition(S, low, high):
13    result = 0
14    pivot_idx = median_of_three(S, low, high)
15    S[low], S[pivot_idx] = S[pivot_idx], S[low]
16    i = low + 1
17    for j in range(low + 1, high, 1):
18        result += 1
19        if S[j] <= S[low]:
20            S[i], S[j] = S[j], S[i]
21            i += 1
22    S[low], S[i - 1] = S[i - 1], S[low]
23    return i - 1, result
24
25 def median_of_three(S, low, high):
26    mid = (low + high - 1) // 2
27    a = S[low]
28    b = S[mid]
29    c = S[high - 1]
30    if a <= b <= c:
31        return b, mid
32    if a <= c <= b:
33        return c, high - 1
34    if b <= c <= a:
35        return c, high - 1
36    return a, low
37
38 listin = [15, 5, 7, 6, 80, 32, 4, 18, 76, 180]
39
40 quickSort(listin)
41 print(listin)
42
```

Nomor 7

Visual Studio Code

no 7.py - Untitled (Workspace) - Visual Studio Code

```
1 from time import time as detik
2 from random import shuffle as kocok
3 import time
4 k = [i for i in range(1,6000)]
5 kocok(k)
6
7 def mergeSort(S):
8     if len(S) <= 1:
9         return S
10    mid = len(S)//2
11    L = S[:mid]
12    R = S[mid:]
13    mergeSort(L)
14    mergeSort(R)
15    i = j = k = 0
16    while i < len(L) and j < len(R):
17        if L[i] < R[j]:
18            S[k] = L[i]
19            i += 1
20        else:
21            S[k] = R[j]
22            j += 1
23        k += 1
24    while i < len(L):
25        S[k] = L[i]
26        i += 1
27        k += 1
28    while j < len(R):
29        S[k] = R[j]
30        j += 1
31        k += 1
32
33 def partition(S, low, high):
34     i = low - 1
35     pivot = S[high]
36     for j in range(low, high):
37         if S[j] < pivot:
38             i += 1
39             S[i], S[j] = S[j], S[i]
40     S[i+1], S[high] = S[high], S[i+1]
41     return (i + 1)
42
43 def quickSort(S, low, high):
44     if low < high:
45         pi = partition(S, low, high)
46         quickSort(S, low, pi-1)
47         quickSort(S, pi+1, high)
48
49 import random
50 def _merge_sort(indices, the_list):
51     start = indices[0]
52     end = indices[1]
53     half_way = (end - start) // 2 + start
54     if start < half_way:
55         _merge_sort((start, half_way), the_list)
56     if half_way + 1 < end and end - start > 1:
57         _merge_sort((half_way + 1, end), the_list)
```

dzakifadh@developer:~/Documents/python\$ python -u "/home/dzakifadh/Documents/python/no 7.py"

merge : 0.0260098 detik
quick : 0.013773 detik
merge mod : -2.40803e-05 detik
quick mod : -0.0250959 detik
dzakifadh@developer:~/Documents/python\$

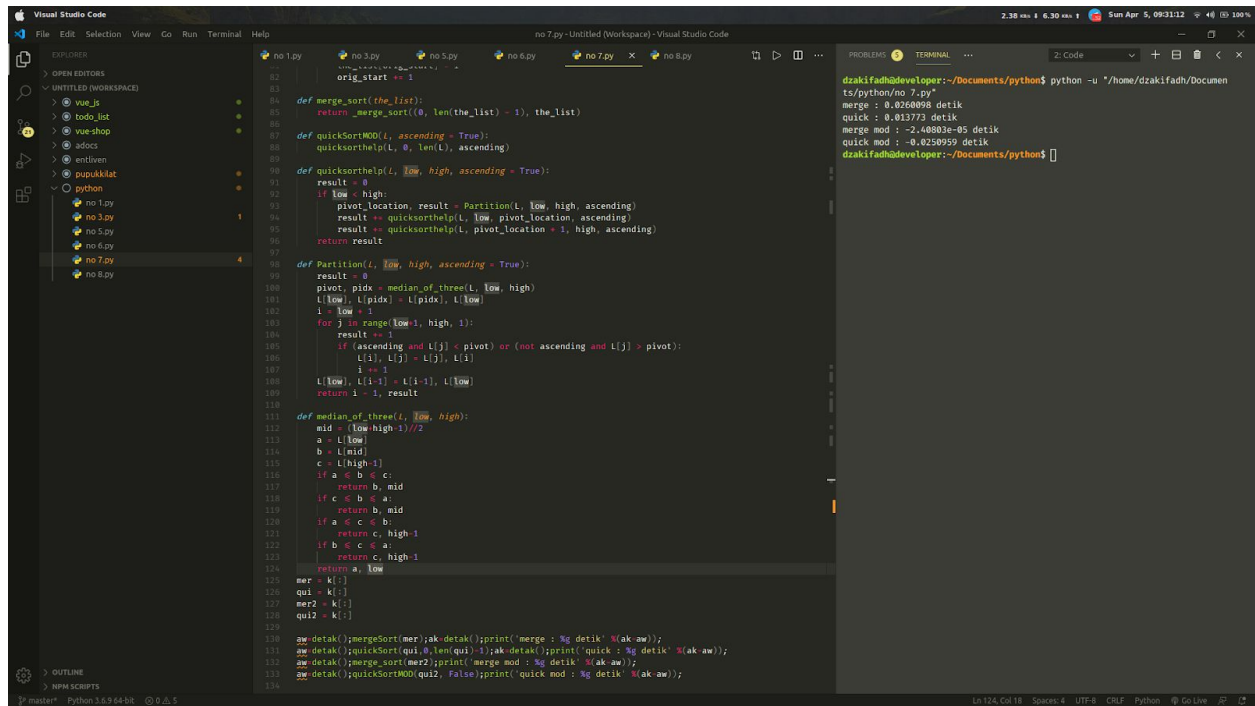
Google Chrome

no 7.py - Untitled (Workspace) - Visual Studio Code

```
46
47 import random
48 def _merge_sort(indices, the_list):
49     start = indices[0]
50     end = indices[1]
51     half_way = (end - start) // 2 + start
52     if start < half_way:
53         _merge_sort((start, half_way), the_list)
54     if half_way + 1 < end and end - start > 1:
55         _merge_sort((half_way + 1, end), the_list)
56
57 sort_sub_list(the_list, indices[0], indices[1])
58
59 def sort_sub_list(the_list, start, end):
60     orig_start = start
61     initial_start_second_list = (end - start) // 2 + start + 1
62     list2_first_index = initial_start_second_list
63     new_list = []
64     while start < initial_start_second_list and list2_first_index < end:
65         first1 = the_list[start]
66         first2 = the_list[list2_first_index]
67         if first1 < first2:
68             new_list.append(first1)
69             list2_first_index += 1
70         else:
71             new_list.append(first2)
72             start += 1
73     while start < initial_start_second_list:
74         new_list.append(the_list[start])
75         start += 1
76
77 while list2_first_index < end:
78     new_list.append(the_list[list2_first_index])
79     list2_first_index += 1
80 for i in new_list:
81     the_list[orig_start] = i
82     orig_start += 1
83
84 def merge_sort(the_list):
85     return _merge_sort((0, len(the_list) - 1), the_list)
86
87 def quickSortMOD(L, ascending = True):
88     quicksorthelp(L, 0, len(L), ascending)
89
90 def quicksorthelp(L, low, high, ascending = True):
91     result = 0
92     if low < high:
93         pivot_location, result = Partition(L, low, high, ascending)
94         result += quicksorthelp(L, low, pivot_location, ascending)
95         result += quicksorthelp(L, pivot_location + 1, high, ascending)
96     return result
97
98 def Partition(L, low, high, ascending = True):
```

dzakifadh@developer:~/Documents/python\$ python -u "/home/dzakifadh/Documents/python/no 7.py"

merge : 0.0260098 detik
quick : 0.013773 detik
merge mod : -2.40803e-05 detik
quick mod : -0.0250959 detik
dzakifadh@developer:~/Documents/python\$



```
no 7.py
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_idx = median_of_three(L, low, high)
    L[low], L[pivot_idx] = L[pivot_idx], 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 = 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) // 2
    a = L[low]
    b = L[mid]
    c = L[high]
    if a <= b & b <= c:
        return b, mid
    if c <= b & b <= a:
        return b, mid
    if a <= c & c <= b:
        return c, high - 1
    if b <= c & c <= a:
        return c, high - 1
    return a, low

mer = k[:]
qui = k[:]
mer2 = k[:]
qui2 = k[:]

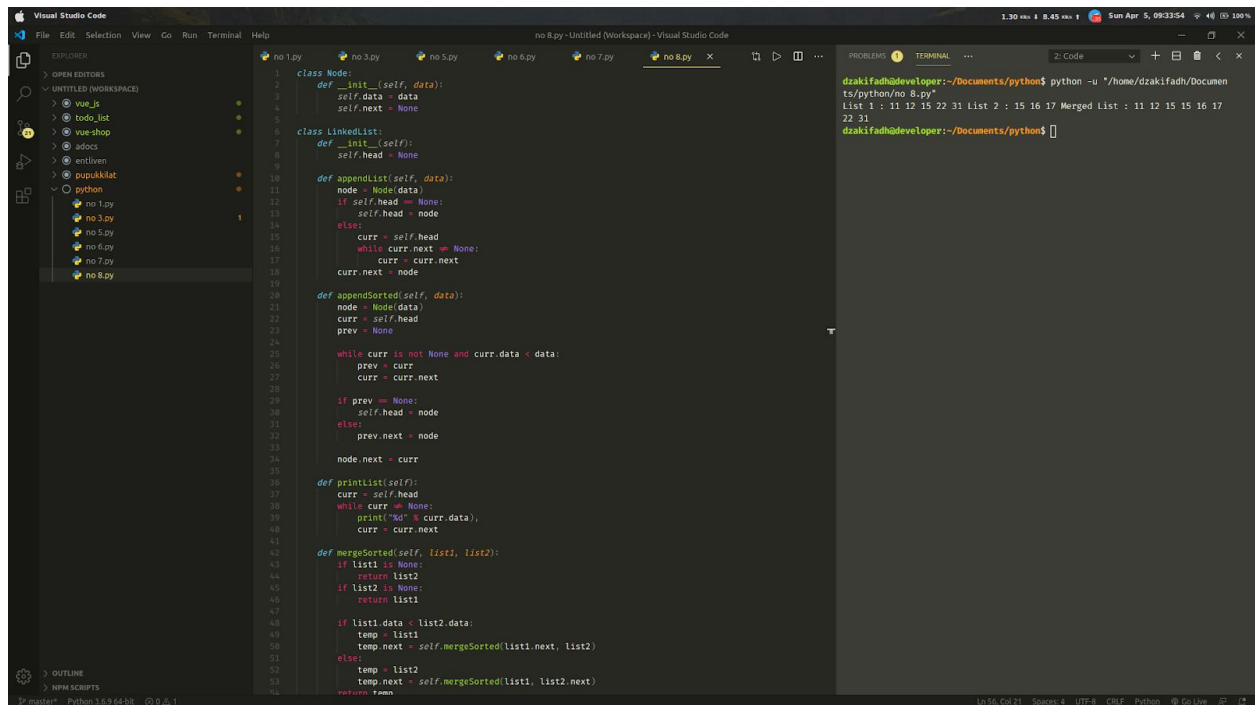
aw-detak();mergeSort(mer);ak-detak();print('merge : %g detik' % (ak-aw));
aw-detak();quickSort(qui,0,len(qui)-1);ak-detak();print('quick : %g detik' % (ak-aw));
aw-detak();merge_sort(mer2);print('merge mod : %g detik' % (ak-aw));
aw-detak();quickSortMOD(qui2, False);print('quick mod : %g detik' % (ak-aw));
```

dzakifadh@Developer:~/Documents/python\$ python -u ~/home/dzakifadh/Documents/python/no 7.py

merge : 0.0268098 detik
quick : 0.013773 detik
merge mod : -2.40803e-05 detik
quick mod : -0.0250959 detik

dzakifadh@Developer:~/Documents/python\$

Nomor 8



```
no 8.py
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)
```

dzakifadh@Developer:~/Documents/python\$ python -u ~/home/dzakifadh/Documents/python/no 8.py

List 1 : 11 12 15 22 31 List 2 : 15 16 17 Merged List : 11 12 15 16 17 22 31

dzakifadh@Developer:~/Documents/python\$

