

**LAPORAN PRAKTIKUM**  
**ALGORITMA DAN STRUKTUR DATA**  
**MODUL 6**  
**PENGATURAN LANJUTAN**

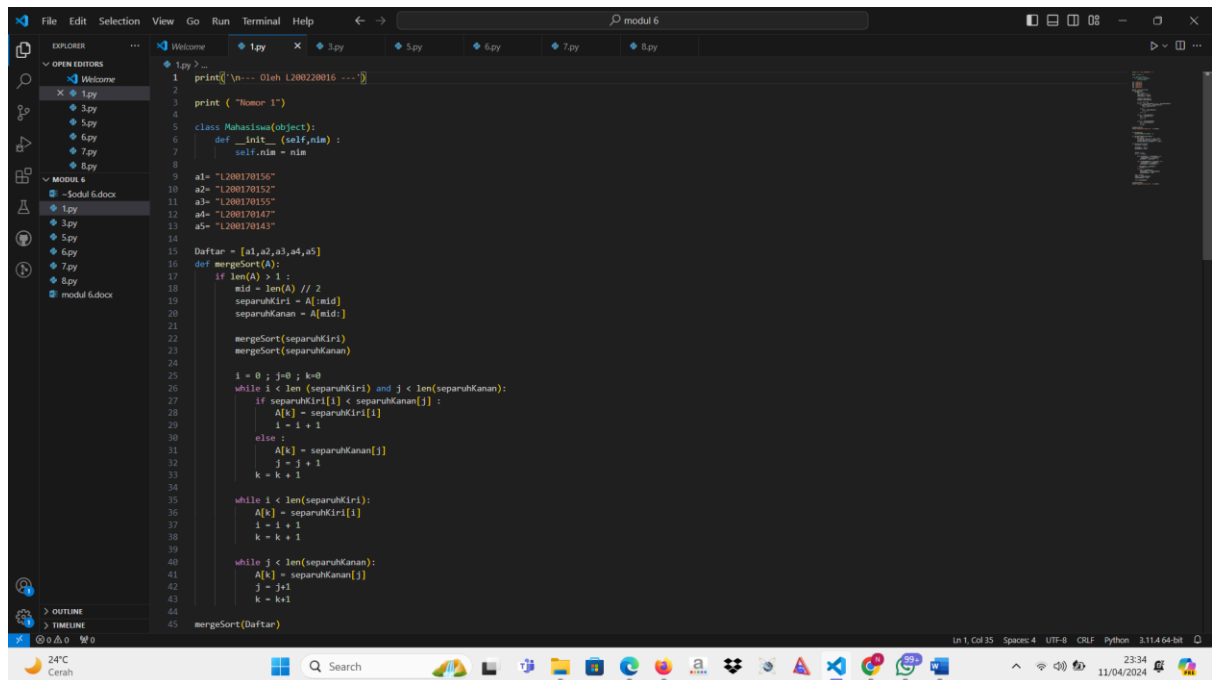


**Disusun oleh:**  
**Nadia Maharani Zulvika**  
**L200220016**  
**A**

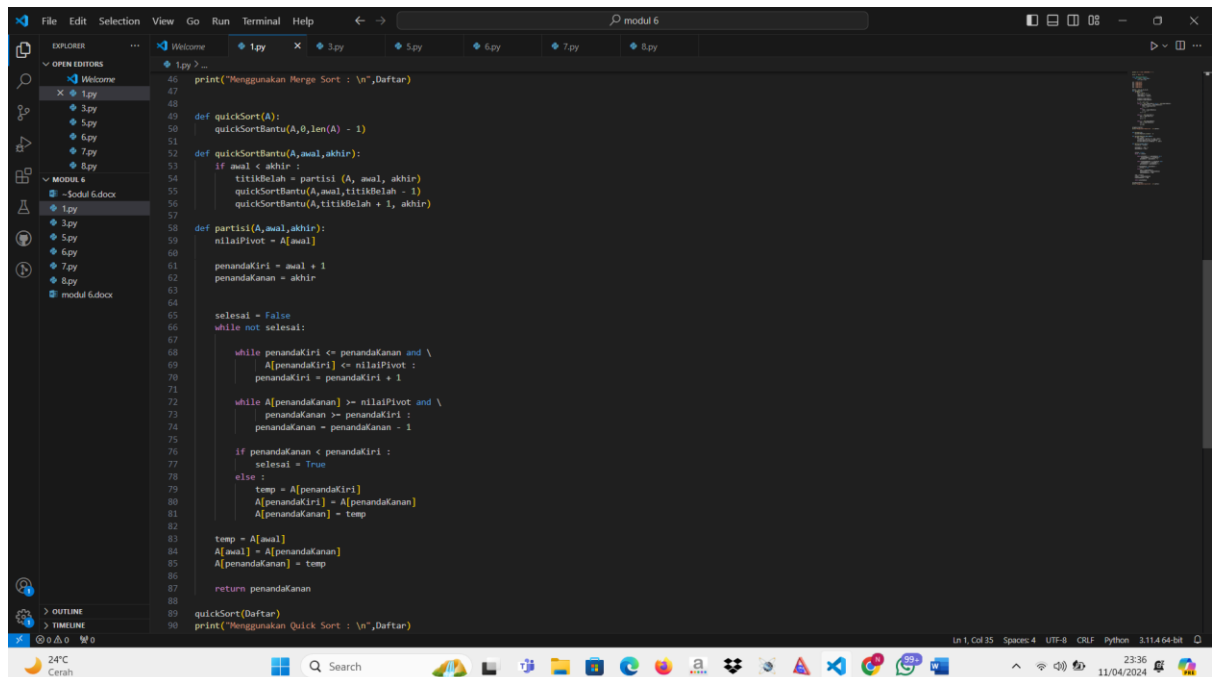
**TEKNIK INFORMATIKA**  
**FAKULTAS KOMUNIKASI DAN INFORMATIKA**  
**UNIVERSITAS MUHAMMADIYAH SURAKARTA**  
**2024**

## Tugas

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.

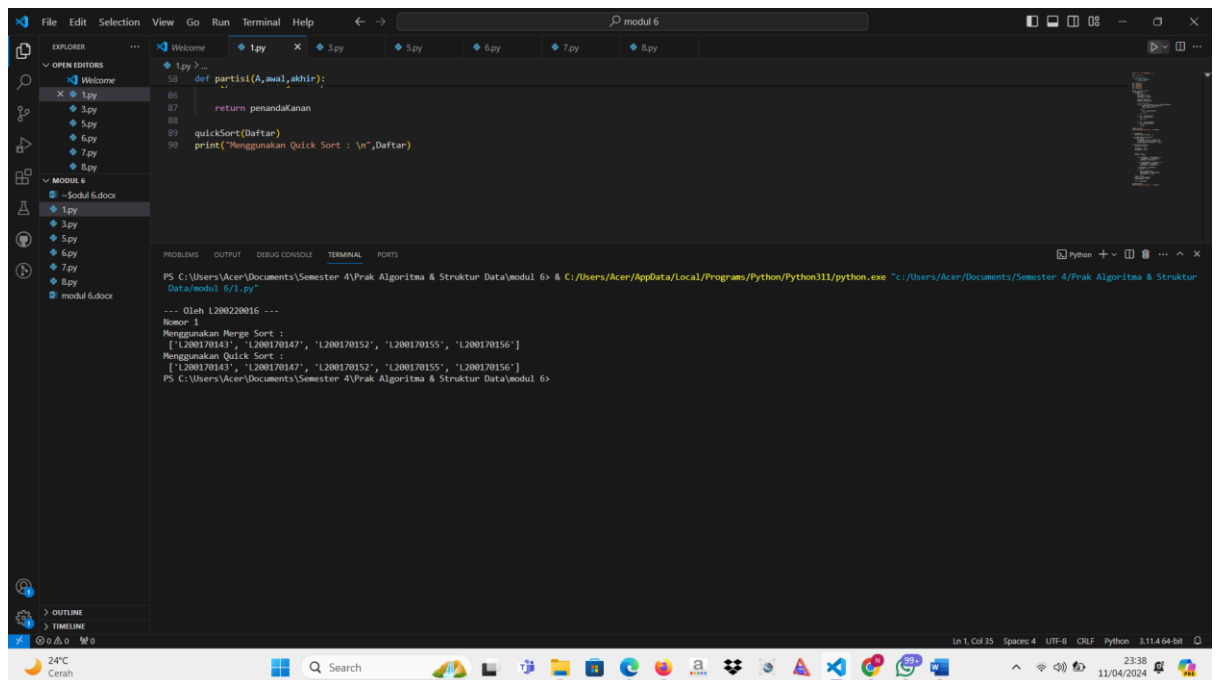


```
1 print("\n--- Oleh L200220016 ---")
2 print("Nomor 1")
3
4 class Mahasiswa(object):
5     def __init__(self, nim):
6         self.nim = nim
7
8 a1= "L200170150"
9 a2= "L200170152"
10 a3= "L200170155"
11 a4= "L200170147"
12 a5= "L200170143"
13
14 Daftar = [a1,a2,a3,a4,a5]
15
16 def mergeSort(A):
17     if len(A) > 1:
18         mid = len(A) // 2
19         separuhKiri = A[:mid]
20         separuhKanan = A[mid:]
21
22         mergeSort(separuhKiri)
23         mergeSort(separuhKanan)
24
25         i = 0; j=0; k=0
26         while i < len(separuhKiri) and j < len(separuhKanan):
27             if separuhKiri[i] < separuhKanan[j]:
28                 A[k] = separuhKiri[i]
29                 i = i + 1
30             else:
31                 A[k] = separuhKanan[j]
32                 j = j + 1
33                 k = k + 1
34
35         while i < len(separuhKiri):
36             A[k] = separuhKiri[i]
37             i = i + 1
38             k = k + 1
39
40         while j < len(separuhKanan):
41             A[k] = separuhKanan[j]
42             j = j + 1
43             k = k + 1
44
45     mergeSort(Daftar)
```



```
46 print("Menggunakan Merge Sort : \n",Daftar)
47
48
49 def quickSort(A):
50     quickSortBantu(A,0,len(A) - 1)
51
52 def quickSortBantu(A,awal,akhir):
53     if awal < akhir:
54         titikBelah = partisi(A, awal, akhir)
55         quickSortBantu(A,awal, titikBelah - 1)
56         quickSortBantu(A, titikBelah + 1, akhir)
57
58 def partisi(A,awal,akhir):
59     nilaiPivot = A[awal]
60
61     penandaKiri = awal + 1
62     penandaKanan = akhir
63
64
65     selesai = False
66     while not selesai:
67
68         while penandaKiri < penandaKanan and \
69             A[penandaKiri] <= nilaiPivot:
70             penandaKiri = penandaKiri + 1
71
72         while A[penandaKanan] >= nilaiPivot and \
73             penandaKanan >= penandaKiri:
74             penandaKanan = penandaKanan - 1
75
76         if penandaKiri < penandaKanan:
77             selesai = True
78         else:
79             temp = A[penandaKiri]
80             A[penandaKiri] = A[penandaKanan]
81             A[penandaKanan] = temp
82
83             temp = A[awal]
84             A[awal] = A[penandaKanan]
85             A[penandaKanan] = temp
86
87     return penandaKanan
88
89 quickSort(Daftar)
90 print("Menggunakan Quick Sort : \n",Daftar)
```

Output:



```
def partisi(A, awal, akhir):
    return penandaKanan

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

```
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6> & C:/Users/Acer/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/Acer/Documents/Semester 4/Prak Algoritma & Struktur Data/modul 6/1.py"

--- Oleh L200220016 ---
Nomor 1
Menggunakan Merge Sort :
['L200170143', 'L200170147', 'L200170152', 'L200170155', 'L200170156']
Menggunakan Quick Sort :
['L200170143', 'L200170147', 'L200170152', 'L200170155', 'L200170156']
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6>
```

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.

(-)

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

```
1 | from time import time as detik
2 | from random import shuffle as kocok
3 | import time
4 | k = range(6000)
5 | kocok(k)
6 | u_bub = k[:] ##
7 | u_sel = k[:] ## Deep copy.
8 | u_ins = k[:] ## Jangan lupa [:]-nya!
9 | u_mrg = k[:] ##
10 | u_qck = k[:] ##
11 |
12 | aw=detak();bubbleSort(u_bub);ak=detak();print('bubble: %g detik' %(ak-aw) );
13 | aw=detak();selectionSort(u_sel);ak=detak();print('selection: %g detik' %(ak-aw) );
14 | aw=detak();insertionSort(u_ins);ak=detak();print('insertion: %g detik' %(ak-aw) );
15 | aw=detak();mergeSort(u_mrg);ak=detak();print('merge: %g detik' %(ak-aw) );
16 | aw=detak();quickSort(u_qck);ak=detak();print('quick: %g detik' %(ak-aw) );
```

Tunjukkan hasil ujinya ke asisten praktikum.

```
1 print("\n--- Oleh L200220016 ---")
2
3 print("Memor 3")
4 from time import time as detik
5 from random import shuffle as kocok
6 import time
7 def swap(A,p,q):
8     tmp = A[p]
9     A[p] = A[q]
10    A[q] = tmp
11
12 def bubbleSort(A):
13     n = len(A)
14     for i in range(n-1):
15         for j in range(n-i-1):
16             if A[j] > A[j+1]:
17                 swap(A,j,j+1)
18
19 def cariPosisiYangTerkecil(A, dariSini, sampaiSini):
20     posisiYangTerkecil = dariSini
21     for i in range(dariSini+1, sampaiSini):
22         if A[i] < A[posisiYangTerkecil]:
23             posisiYangTerkecil = i
24     return posisiYangTerkecil
25
26 def selectionSort(A):
27     n = len(A)
28     for i in range(n-1):
29         indexKecil = cariPosisiYangTerkecil(A, i, n)
30         if indexKecil != i:
31             swap(A, i, indexKecil)
32
33 def insertionSort(A):
34     n = len(A)
35     for i in range(1, n):
36         nilai = A[i]
37         pos = i
38         while pos > 0 and nilai < A[pos - 1]:
39             A[pos] = A[pos - 1]
40             pos = pos - 1
41         A[pos] = nilai
42
43 def mergeSort(A):
44     if len(A) > 1:
45         mid = len(A) // 2
```

```
43 def mergeSort(A):
44     if len(A) > 1:
45         mid = len(A) // 2
46         separuhKiri = A[:mid]
47         separuhKanan = A[mid:]
48
49         mergeSort(separuhKiri)
50         mergeSort(separuhKanan)
51
52         i = 0; j = 0; k = 0
53         while i < len(separuhKiri) and j < len(separuhKanan):
54             if separuhKiri[i] < separuhKanan[j]:
55                 A[k] = separuhKiri[i]
56                 i = i + 1
57             else:
58                 A[k] = separuhKanan[j]
59                 j = j + 1
60             k = k + 1
61
62         while i < len(separuhKiri):
63             A[k] = separuhKiri[i]
64             i = i + 1
65             k = k + 1
66
67         while j < len(separuhKanan):
68             A[k] = separuhKanan[j]
69             j = j + 1
70             k = k + 1
71
72 def quickSort(A):
73     quickSortBantu(A, 0, len(A) - 1)
74
75 def quickSortBantu(A, awal, akhir):
76     if awal < akhir:
77         titikBelah = partisi(A, awal, akhir)
78         quickSortBantu(A, awal, titikBelah - 1)
79         quickSortBantu(A, titikBelah + 1, akhir)
80
81 def partisi(A, awal, akhir):
82     nilaiPivot = A[awal]
83     penandaKiri = awal + 1
84     penandaKanan = akhir
85
86     selesai = False
87     while not selesai:
```

```

87 while not selesai:
88     while penandaKiri <= penandaKanan and \
89         A[penandaKiri] <= nilaiPivot :
90         penandaKiri = penandaKiri + 1
91
92     while A[penandaKanan] >= nilaiPivot and \
93         penandaKanan >= penandaKiri :
94         penandaKanan = penandaKanan - 1
95
96     if penandaKanan < penandaKiri :
97         selesai = True
98     else :
99         temp = A[penandaKiri]
100         A[penandaKiri] = A[penandaKanan]
101         A[penandaKanan] = temp
102
103     return penandaKanan
104
105 k = []
106 for i in range(1, 6001):
107     k.append(i)
108     kocok(k)
109
110 u_bub = k[:]
111 u_sel = k[:]
112 u_ins = k[:]
113 u_mrg = k[:]
114 u_qck = k[:]
115
116 aw = detak();bubbleSort(u_bub);ak-detak();print("bubble : %g detik" %(ak-aw));
117 aw = detak();selectionSort(u_sel);ak-detak();print("selection: %g detik" %(ak-aw));
118 aw = detak();insertionSort(u_ins);ak-detak();print("insertion: %g detik" %(ak-aw));
119 aw = detak();mergeSort(u_mrg);ak-detak();print("merge: %g detik" %(ak-aw));
120 aw = detak();quickSort(u_qck);ak-detak();print("quick : %g detik" %(ak-aw));
121
122
123
124
125
126

```

Output:

```

124 aw = detak();mergeSort(u_mrg);ak-detak();print("merge: %g detik" %(ak-aw));
125 aw = detak();quickSort(u_qck);ak-detak();print("quick : %g detik" %(ak-aw));
126

```

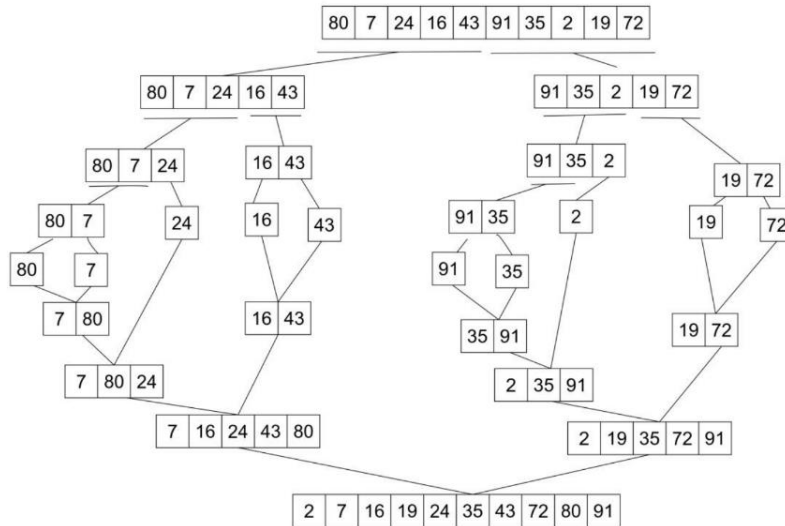
```

PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6> & C:/Users/Acer/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/Acer/Documents/Semester 4/Prak Algoritma & Struktur Data/modul 6/3.py"

--- Oleh L200220016 ---
Runer 3
bubble : 1.04684 detik
selection: 0.706403 detik
insertion : 0.851178 detik
merge: 0.0115994 detik
quick : 0.00718784 detik
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6>

```

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



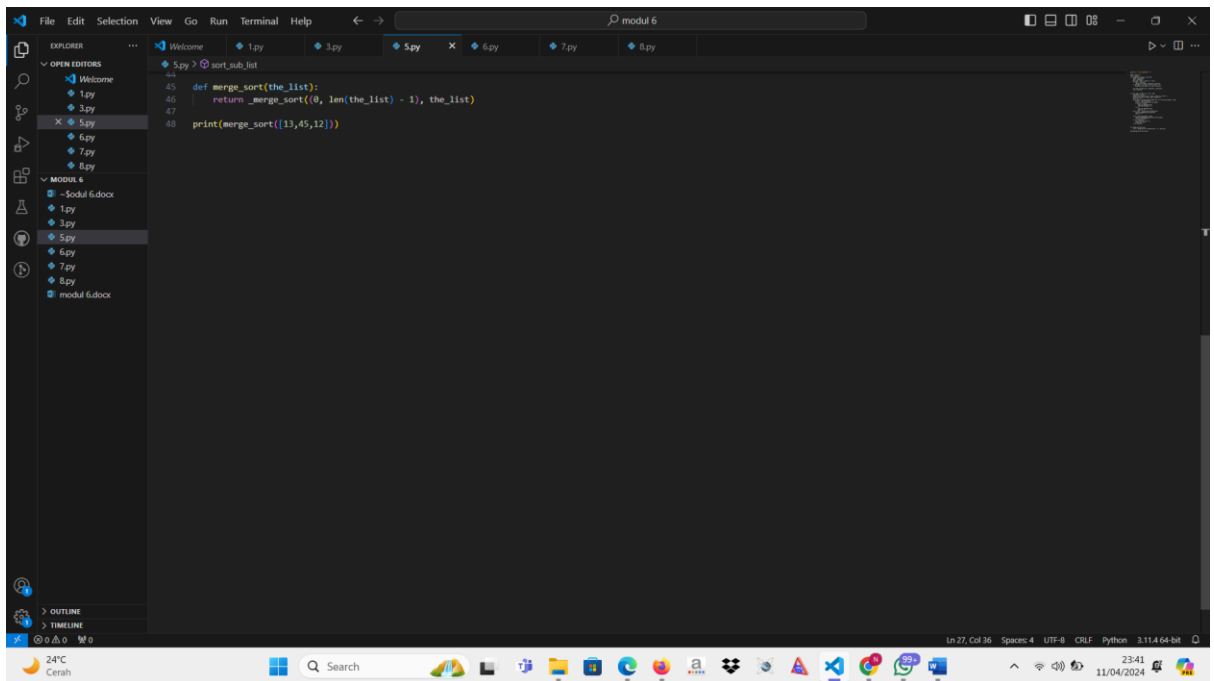
(b) quick sort

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.

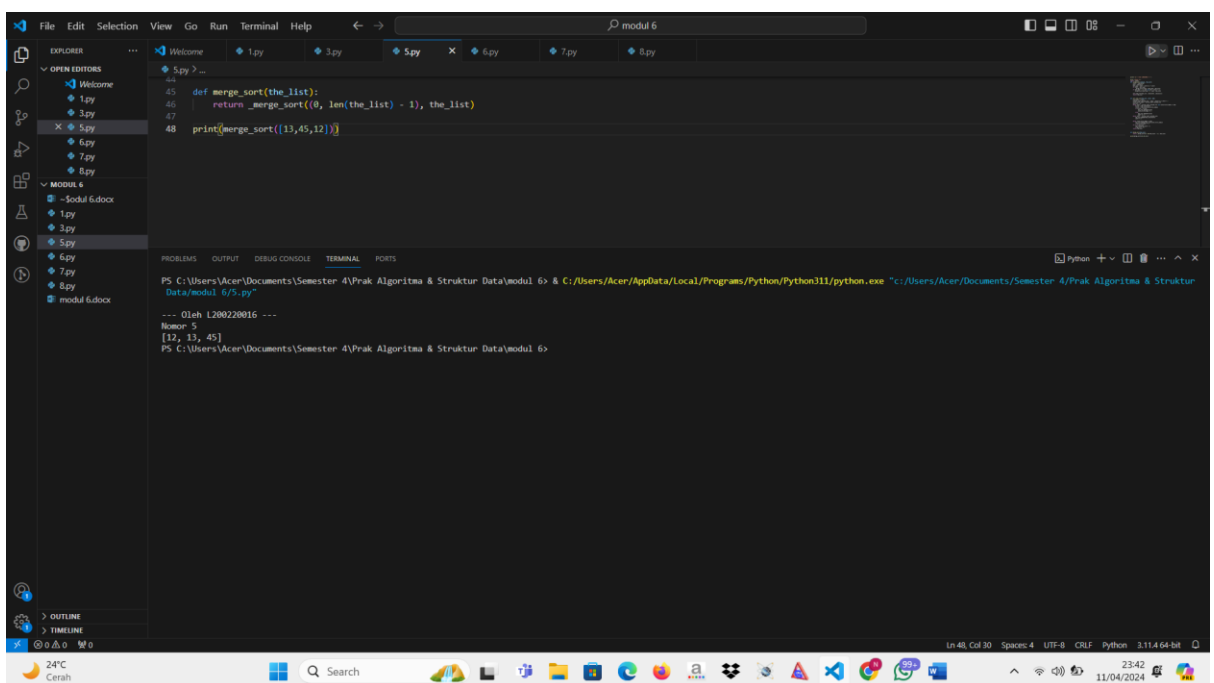
```

1 print("\n--- Olee L200220016 ---")
2
3 print("Memor 5")
4 import random
5
6 def _merge_sort(indices, the_list):
7     start = indices[0]
8     end = indices[1]
9     half_way = (end - start) // 2 + start
10    if start < half_way:
11        _merge_sort((start, half_way), the_list)
12    if half_way + 1 <= end and end - start != 1:
13        _merge_sort((half_way + 1, end), the_list)
14    sort_sub_list(the_list, indices[0], indices[1])
15    return the_list
16
17
18 def sort_sub_list(the_list, start, end):
19     orig_start = start
20     initial_start_second_list = (end - start) // 2 + start + 1
21     list2_first_index = initial_start_second_list
22     new_list = []
23     while start < initial_start_second_list and list2_first_index <= end:
24         first1 = the_list[start]
25         first2 = the_list[list2_first_index]
26         if first1 > first2:
27             new_list.append(first2)
28             list2_first_index += 1
29         else:
30             new_list.append(first1)
31             start += 1
32     while start < initial_start_second_list:
33         new_list.append(the_list[start])
34         start += 1
35     while list2_first_index <= end:
36         new_list.append(the_list[list2_first_index])
37         list2_first_index += 1
38     for i in new_list:
39         the_list[orig_start] = i
40         orig_start += 1
41     return the_list
42
43
44
45 def merge_sort(the_list):

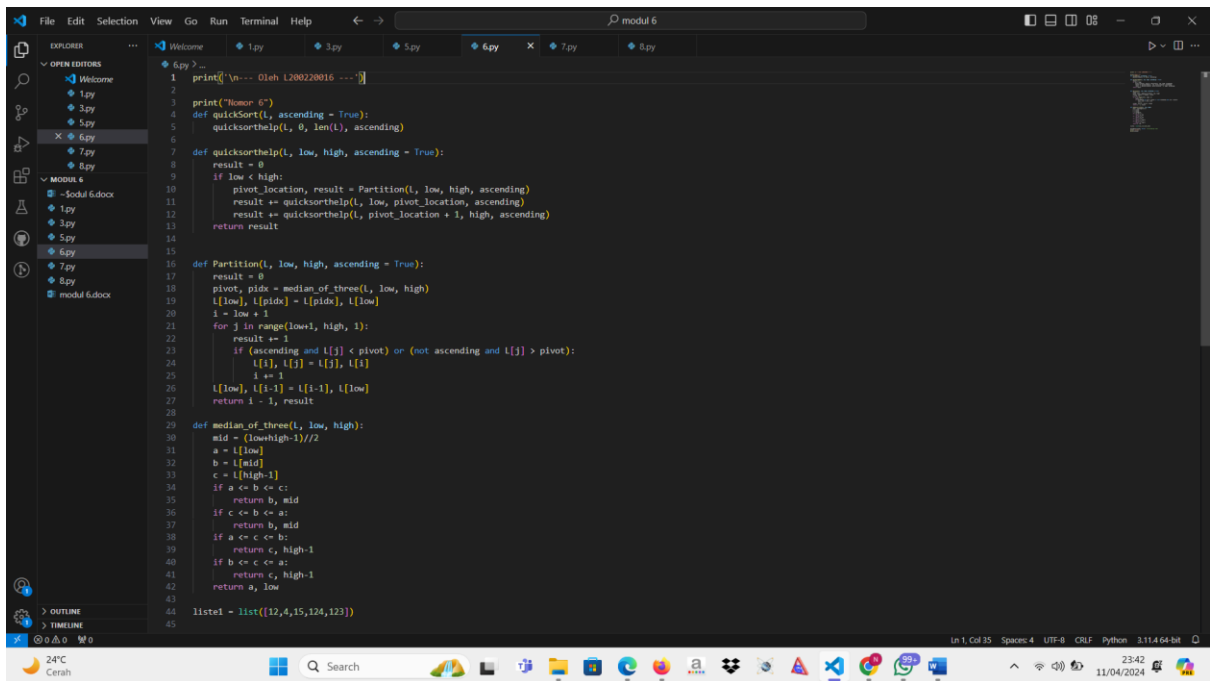
```



Output:

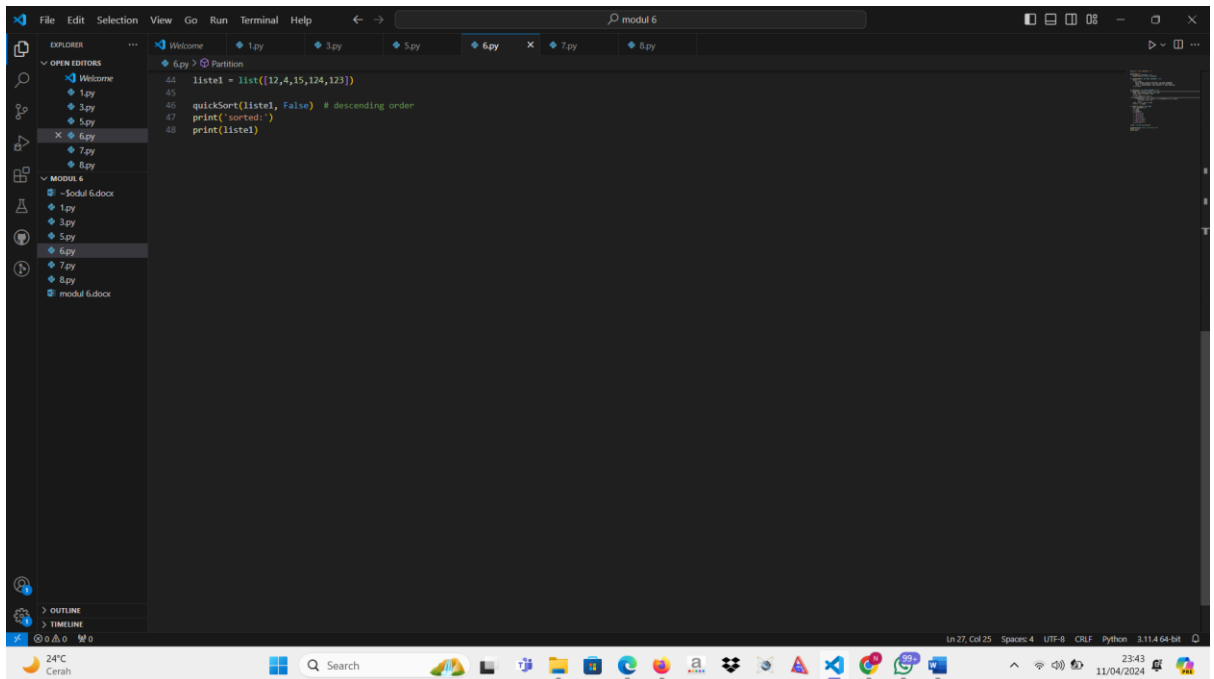


6. Apakah kita bisa meningkatkan efisiensi program quickSort dengan memakai metode median-dari-tiga untuk memilih pivotnya? Ubahlah kodenya dan ujilah.



This screenshot shows the Visual Studio Code editor with a Python file named 'modul 6'. The code implements a quicksort algorithm. It starts with a print statement, followed by a recursive function 'quicksort' and a helper function 'quicksorthelp'. The 'Partition' function is used to find the pivot and rearrange the array. The 'median\_of\_three' function is used to select the pivot. The main execution line is 'list1 = list([12,4,15,124,123])'.

```
1 print('\n--- Oieh 1200220016 ---\n')
2
3 print("Memor 6")
4 def quicksort(l, ascending = True):
5     quicksorthelp(l, 0, len(l), ascending)
6
7 def quicksorthelp(l, low, high, ascending = True):
8     result = 0
9     if low < high:
10        pivot_location, result = Partition(l, low, high, ascending)
11        result += quicksorthelp(l, low, pivot_location, ascending)
12        result += quicksorthelp(l, pivot_location + 1, high, ascending)
13    return result
14
15 def Partition(l, low, high, ascending = True):
16    result = 0
17    pivot, pidx = median_of_three(l, low, high)
18    l[low], l[pidx] = l[pidx], l[low]
19    i = low + 1
20    for j in range(low+1, high, 1):
21        result += 1
22        if (ascending and l[j] < pivot) or (not ascending and l[j] > pivot):
23            l[i], l[j] = l[j], l[i]
24            i += 1
25    l[low], l[i-1] = l[i-1], l[low]
26    return i - 1, result
27
28 def median_of_three(l, low, high):
29    mid = (low+high-1)//2
30    a = l[low]
31    b = l[mid]
32    c = l[high-1]
33    if a <= b <= c:
34        return b, mid
35    if c <= b <= a:
36        return b, mid
37    if a <= c <= b:
38        return c, high-1
39    if b <= c <= a:
40        return c, high-1
41    return c, low
42
43 list1 = list([12,4,15,124,123])
44
45
```



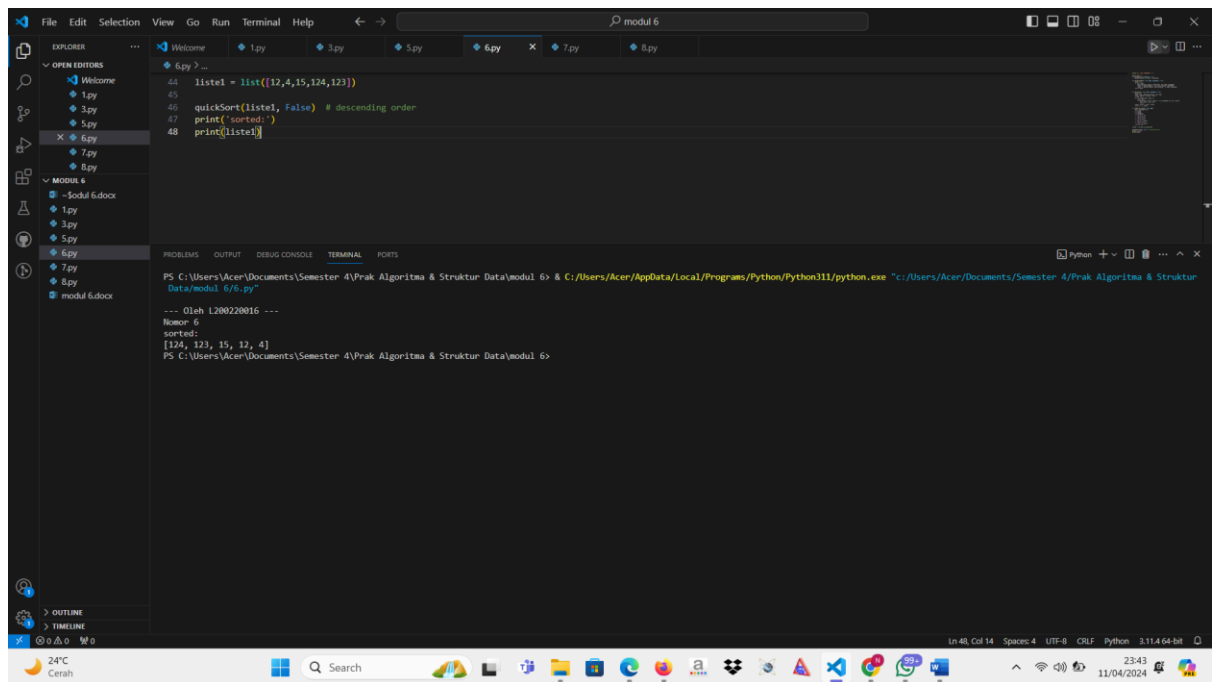
This screenshot shows the Visual Studio Code editor with the same Python file 'modul 6'. The code is now executed, and the output is displayed in the terminal. The output shows the list [12, 4, 15, 124, 123] being sorted in descending order, resulting in [124, 123, 15, 4, 12].

```
44 list1 = list([12,4,15,124,123])
45
46 quicksort(list1, False) # descending order
47 print('sorted:')
48 print(list1)
```

Ln 27, Col 25 Spaces: 4 UTF-8 CRLF Python 3.11.4 64-bit



Output:



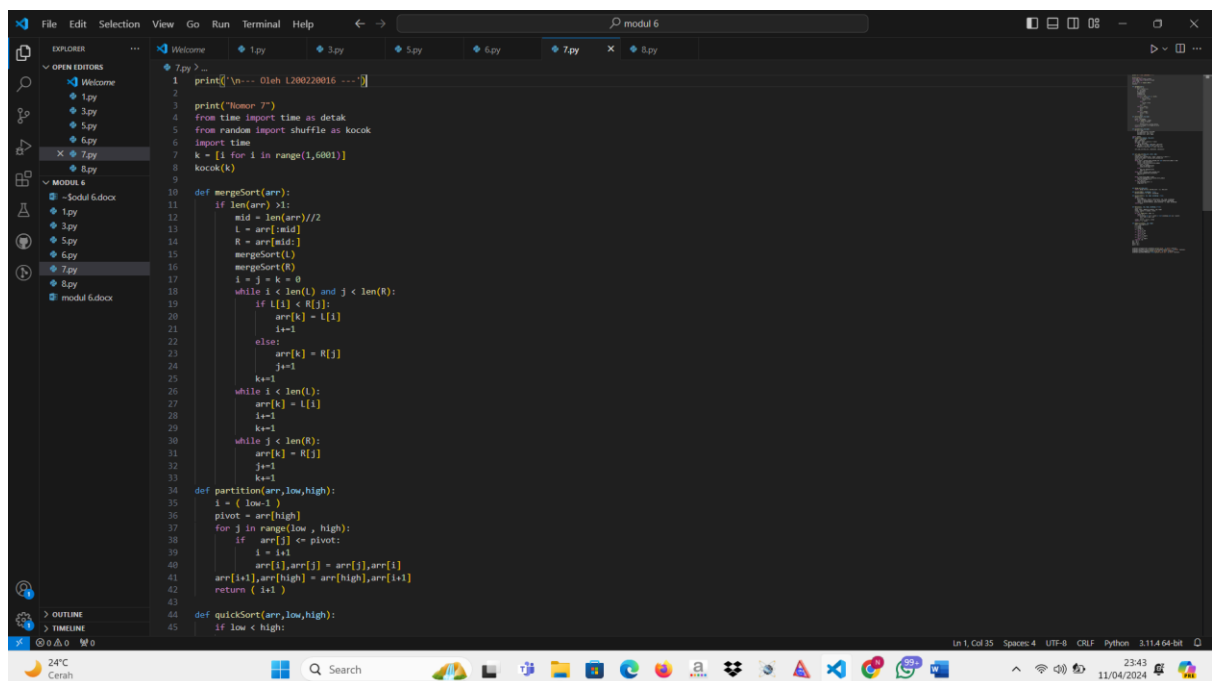
The screenshot shows a VS Code editor with a Python file named 'modul 6'. The code in the editor is as follows:

```
44 listel = list([12,4,15,124,123])
45
46 quickSort(listel, False) # descending order
47 print('sorted:')
48 print(listel)
```

The terminal output shows the execution of the code:

```
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6> C:\Users\Acer\AppData\Local\Programs\Python\Python311\python.exe "c:/Users/Acer/Documents/Semester 4/Prak Algoritma & Struktur Data/modul 6/6.py"
--- Oleh L200220016 ---
Number: 6
sorted:
[124, 123, 15, 12, 4]
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6>
```

7. Uji-kecepatan keduanya dan perbandingan juga dengan kode awalnya.



The screenshot shows a VS Code editor with a Python file named 'modul 6'. The code in the editor is as follows:

```
1 print('\n--- Oleh L200220016 ---')
2
3 print("Homer 7")
4 from time import time as detik
5 from random import shuffle as kocok
6 import time
7 k = [1 for i in range(1,6001)]
8 kocok(k)
9
10 def mergeSort(arr):
11     if len(arr) > 1:
12         mid = len(arr)//2
13         l = arr[:mid]
14         R = arr[mid:]
15         mergeSort(l)
16         mergeSort(R)
17         i = j = k = 0
18         while i < len(L) and j < len(R):
19             if L[i] < R[j]:
20                 arr[k] = L[i]
21                 i+=1
22             else:
23                 arr[k] = R[j]
24                 j+=1
25             k+=1
26         while i < len(L):
27             arr[k] = L[i]
28             i+=1
29             k+=1
30         while j < len(R):
31             arr[k] = R[j]
32             j+=1
33             k+=1
34
35 def partition(arr,low,high):
36     i = (low-1)
37     pivot = arr[high]
38     for j in range(low , high):
39         if arr[j] <= pivot:
40             i = i+1
41             arr[i],arr[j] = arr[j],arr[i]
42     arr[i+1],arr[high] = arr[high],arr[i+1]
43     return ( i+1 )
44
45 def quickSort(arr,low,high):
46     if low < high:
```

```
44 def quickSort(arr, low, high):
45     if low < high:
46         pi = partition(arr, low, high)
47         quickSort(arr, low, pi-1)
48         quickSort(arr, pi+1, high)
49
50 import random
51
52 def _merge_sort(indices, the_list):
53     start = indices[0]
54     end = indices[1]
55     half_way = (end - start)//2 + start
56     if start < half_way:
57         _merge_sort(start, half_way, the_list)
58     if half_way + 1 <= end and end - start != 1:
59         _merge_sort(half_way + 1, end, the_list)
60
61     sort_sub_list(the_list, indices[0], indices[1])
62
63
64 def sort_sub_list(the_list, start, end):
65     orig_start = start
66     initial_start_second_list = (end - start)//2 + start + 1
67     list2_first_index = initial_start_second_list
68     new_list = []
69     while start < initial_start_second_list and list2_first_index <= end:
70         first1 = the_list[start]
71         first2 = the_list[list2_first_index]
72         if first1 > first2:
73             new_list.append(first2)
74             list2_first_index += 1
75         else:
76             new_list.append(first1)
77             start += 1
78     while start < initial_start_second_list:
79         new_list.append(the_list[start])
80         start += 1
81
82     while list2_first_index <= end:
83         new_list.append(the_list[list2_first_index])
84         list2_first_index += 1
85     for i in new_list:
86         the_list[orig_start] = i
87         orig_start += 1
88
```

```
91 def merge_sort(the_list):
92     return _merge_sort(0, len(the_list) - 1, the_list)
93
94 def quickSortMOD(l, ascending = True):
95     quicksorthelp(l, 0, len(l), ascending)
96
97 def quicksorthelp(l, low, high, ascending = True):
98     result = 0
99     if low < high:
100         pivot_location, result = Partition(l, low, high, ascending)
101         result += quicksorthelp(l, low, pivot_location, ascending)
102         result += quicksorthelp(l, pivot_location + 1, high, ascending)
103     return result
104
105
106 def Partition(l, low, high, ascending = True):
107     result = 0
108     pivot, pidx = median_of_three(l, low, high)
109     l[low], l[pidx] = l[pidx], l[low]
110     i = low + 1
111     for j in range(low+1, high, 1):
112         result += 1
113         if (ascending and l[j] < pivot) or (not ascending and l[j] > pivot):
114             l[i], l[j] = l[j], l[i]
115             i += 1
116     l[low], l[i-1] = l[i-1], l[low]
117     return i - 1, result
118
119 def median_of_three(l, low, high):
120     mid = (low+high-1)//2
121     a = l[low]
122     b = l[mid]
123     c = l[high-1]
124     if a <= b <= c:
125         return b, mid
126     if c <= b <= a:
127         return b, mid
128     if a <= c <= b:
129         return c, high-1
130     if b <= c <= a:
131         return c, high-1
132     return a, low
133
134 mer = k[:]
135 qui = k[:]
```

```
133 mer = k[:]
134 qui = k[:]
135 mer2 = k[:]
136 qui2 = k[:]
137
138
139 awdetak();mergeSort(mer);ak=detak();print('merge : %g detik' %(ak-aw));
140 awdetak();quickSort(qui,0,len(qui)-1);ak=detak();print('quick : %g detik' %(ak-aw));
141 awdetak();merge_sort(mer2);print('merge mod : %g detik' %(ak-aw));
142 awdetak();quickSortMOD(qui2, False);print('quick mod : %g detik' %(ak-aw));
```

Output:

```
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6> C:\Users\Acer\AppData\Local\Programs\Python\Python311\python.exe "c:/Users/Acer/Documents/Semester 4/Prak Algoritma & Struktur Data/modul 6/7.py"

--- Olsh 1200220016 ---
Homor 7
merge : 0.8123608 detik
quick : 0.00891852 detik
merge mod : 0 detik
quick mod : -0.8200274 detik
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6>
```

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

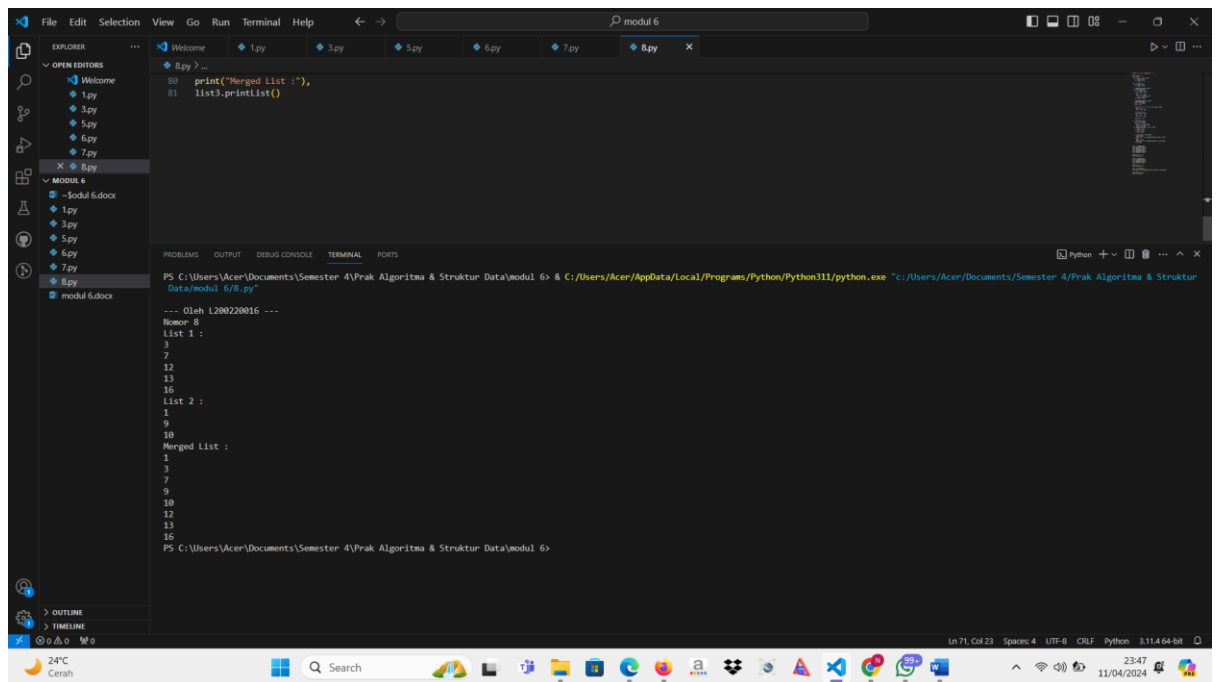
This screenshot shows the first 45 lines of a Python script in VS Code. The script defines a `Node` class and a `LinkedList` class. The `Node` class has an `__init__` method that takes `data` and sets `self.data` and `self.next` to `None`. The `LinkedList` class has an `__init__` method that sets `self.head` to `None`. It also has methods `appendList`, `appendSorted`, `printList`, and `mergeSorted`. The `appendList` method adds a new node to the end of the list. The `appendSorted` method inserts a new node into the list at the correct position to maintain sorted order. The `printList` method prints the data of all nodes in the list. The `mergeSorted` method merges two sorted linked lists into a new sorted linked list.

```
1 print("\n--- Olieh L200220016 ---")
2
3 print("Memor 8")
4 class Node:
5     def __init__(self, data):
6         self.data = data
7         self.next = None
8
9 class LinkedList:
10     def __init__(self):
11         self.head = None
12
13     def appendList(self, data):
14         node = Node(data)
15         if self.head == None:
16             self.head = node
17         else:
18             curr = self.head
19             while curr.next != None:
20                 curr = curr.next
21             curr.next = node
22
23     def appendSorted(self, data):
24         node = Node(data)
25         curr = self.head
26         prev = None
27
28         while curr is not None and curr.data < data:
29             prev = curr
30             curr = curr.next
31
32         if prev == None:
33             self.head = node
34         else:
35             prev.next = node
36             node.next = curr
37
38     def printList(self):
39         curr = self.head
40         while curr != None:
41             print("%d "%curr.data),
42             curr = curr.next
43
44     def mergeSorted(self, list1, list2):
45         if list1 is None:
```

This screenshot shows the continuation of the Python script from the previous screenshot, lines 46 to 81. It completes the `mergeSorted` method and then creates two sorted linked lists, `list1` and `list2`, and merges them into a new sorted linked list `list3`. The `mergeSorted` method returns the head of the merged list. The script then prints the data of all nodes in `list1`, `list2`, and the merged `list3`.

```
46         if list1 is None:
47             return list2
48         if list2 is None:
49             return list1
50
51         if list1.data < list2.data:
52             temp = list1
53             temp.next = self.mergeSorted(list1.next, list2)
54         else:
55             temp = list2
56             temp.next = self.mergeSorted(list1, list2.next)
57         return temp
58
59 list1 = LinkedList()
60 list1.appendSorted(11)
61 list1.appendSorted(12)
62 list1.appendSorted(3)
63 list1.appendSorted(16)
64 list1.appendSorted(7)
65
66 print("List 1 :"),
67 list1.printList()
68
69 list2 = LinkedList()
70 list2.appendSorted(9)
71 list2.appendSorted(10)
72 list2.appendSorted(1)
73
74 print("List 2 :"),
75 list2.printList()
76
77 list3 = LinkedList()
78 list3.head = list3.mergeSorted(list1.head, list2.head)
79
80 print("Merged List :"),
81 list3.printList()
```

Output:



The screenshot shows a Visual Studio Code editor window with a Python file named 'modul 6'. The code in the editor is as follows:

```
80 print("Merged List :"),
81 list3.printlist()
```

The terminal output shows the execution of the script, displaying the merged list and the list3 contents.

```
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6> & C:/Users/Acer/AppData/Local/Programs/Python/Python311/python.exe "c:/Users/Acer/Documents/Semester 4/Prak Algoritma & Struktur Data/modul 6/8.py"

--- Oleh L200220016 ---
Number : 8
List 1 :
3
7
12
13
16
List 2 :
1
9
10
Merged List :
1
3
7
9
10
12
13
16
PS C:\Users\Acer\Documents\Semester 4\Prak Algoritma & Struktur Data\modul 6>
```

The terminal output also shows the file path and the command used to run the script.