

PRACTICUM REPORT
ALGORITHM AND DATA STRUCTURES MODUL
6 : ADVANCED SEQUENCE



Disusun Oleh :
ONIC AGUSTINO
L200234275
X

INFORMATICS ENGINEERING
FACULTY OF COMMUNICATION AND INFORMATICS
UNIVERSITAS MUHAMMADIYAH SURAKARTA
YEARS 2024/2025

TASK:

```
data = [  
    MhsTIF('Ika', 10, 'Sukoharjo', 240000),  
    MhsTIF('Budi', 51, 'Sragen', 230000),  
    MhsTIF('Ahmad', 2, 'Surakarta', 250000),  
    MhsTIF('Chandra', 18, 'Surakarta', 235000),  
    MhsTIF('Eka', 4, 'Boyolali', 240000)  
]
```

Picture 1.1

1.

```
1  #nomer 1
2  class MhsTIF:
3      def __init__(self, nama, NIM, kota, uangSaku):
4          self.nama = nama
5          self.NIM = NIM
6          self.kota = kota
7          self.uangSaku = uangSaku
8
9      def __repr__(self):
10         return f'{self.nama} (NIM: {self.NIM})'
11
12  # Merge Sort untuk objek MhsTIF berdasarkan NIM
13  def mergeSort(arr):
14      if len(arr) > 1:
15          mid = len(arr) // 2
16          left_half = arr[:mid]
17          right_half = arr[mid:]
18
19          mergeSort(left_half)
20          mergeSort(right_half)
21
22          i = j = k = 0
23
24          while i < len(left_half) and j < len(right_half):
25              if left_half[i].NIM < right_half[j].NIM:
26                  arr[k] = left_half[i]
27                  i += 1
28              else:
29                  arr[k] = right_half[j]
30                  j += 1
31              k += 1
32
33          while i < len(left_half):
34              arr[k] = left_half[i]
35              i += 1
36              k += 1
37
38          while j < len(right_half):
39              arr[k] = right_half[j]
40              j += 1
41              k += 1
42
43  # Quick Sort untuk objek MhsTIF berdasarkan NIM
44  def partition(arr, low, high):
45      pivot = arr[high].NIM
46      i = low - 1
47      for j in range(low, high):
48          if arr[j].NIM <= pivot:
49              i += 1
50              arr[i], arr[j] = arr[j], arr[i]
51      arr[i + 1], arr[high] = arr[high], arr[i + 1]
52      return i + 1
53
54  def quickSort(arr, low, high):
55      if low < high:
56          pi = partition(arr, low, high)
57          quickSort(arr, low, pi - 1)
58          quickSort(arr, pi + 1, high)
59
60
61  print('Sebelum Sort:', data)
62  mergeSort(data)
63  print('Setelah Merge Sort:', data)
64
65  quickSort(data, 0, len(data) - 1)
66  print('Setelah Quick Sort:', data)
67
68
```

Picture 1.2

This code defines a **MhsTIF** class to store student data, then implements Merge Sort and Quick Sort algorithms to sort a list of students by NIM , and tests their performance using random data.

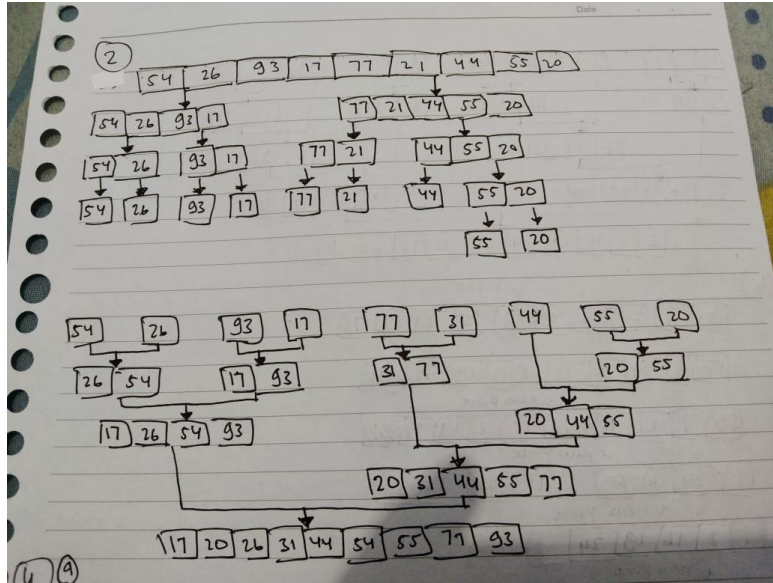


Sebelum Sort: [Ika (NIM: 10), Budi (NIM: 51), Ahmad (NIM: 2), Chandra (NIM: 18), Eka (NIM: 4)]

Setelah Merge Sort: [Ahmad (NIM: 2), Eka (NIM: 4), Ika (NIM: 10), Chandra (NIM: 18), Budi (NIM: 51)]

Setelah Quick Sort: [Ahmad (NIM: 2), Eka (NIM: 4), Ika (NIM: 10), Chandra (NIM: 18), Budi (NIM: 51)]

Picture 1.3 2.



Picture 1.4

3.

```
1 #nomer 3
2 import time
3 from random import shuffle as kocok
4
5 class MhsTIF:
6     def __init__(self, nama, NIM, kota, uangSaku):
7         self.nama = nama
8         self.NIM = NIM
9         self.kota = kota
10        self.uangSaku = uangSaku
11
12    def __repr__(self):
13        return f'{self.nama} (NIM: {self.NIM})'
14
15
16 # Merge Sort
17 def mergeSort(arr):
18     if len(arr) > 1:
19         mid = len(arr) // 2
20         left_half = arr[:mid]
21         right_half = arr[mid:]
22
23         mergeSort(left_half)
24         mergeSort(right_half)
25
26         i = j = k = 0
27
28         while i < len(left_half) and j < len(right_half):
29             if left_half[i].NIM < right_half[j].NIM:
30                 arr[k] = left_half[i]
31                 i += 1
32             else:
33                 arr[k] = right_half[j]
34                 j += 1
35                 k += 1
36
37         while i < len(left_half):
38             arr[k] = left_half[i]
39             i += 1
40             k += 1
41
42         while j < len(right_half):
43             arr[k] = right_half[j]
44             j += 1
45             k += 1
46
47 # Quick Sort
48 def partition(arr, low, high):
49     pivot = arr[high].NIM
50     i = low - 1
51     for j in range(low, high):
52         if arr[j].NIM <= pivot:
53             i += 1
54             arr[i], arr[j] = arr[j], arr[i]
55     arr[i + 1], arr[high] = arr[high], arr[i + 1]
56     return i + 1
57
58 def quickSort(arr, low, high):
59     if low < high:
60         pi = partition(arr, low, high)
61         quickSort(arr, low, pi - 1)
62         quickSort(arr, pi + 1, high)
63
64 # Buat data MhsTIF secara acak
65 def generate_data(n):
66     names = ['Ika', 'Budi', 'Ahmad', 'Chandra', 'Eka', 'Fandi', 'Deni', 'Galuh', 'Janto', 'Hasan']
67     data = [MhsTIF(names[i % len(names)], i, 'Kota ' + str(i), 240000 + i * 1000) for i in range(n)]
68     kocok(data)
69     return data
70
71 # Tes kecepatan
72 data = generate_data(6000)
73 u_mrg = data[:]
74 u_qck = data[:]
75
76 aw = time.time(); mergeSort(u_mrg); ak = time.time(); print('merge: %g detik' % (ak - aw))
77 aw = time.time(); quickSort(u_qck, 0, len(u_qck) - 1); ak = time.time(); print('quick: %g detik' %
78 (ak - aw))
```

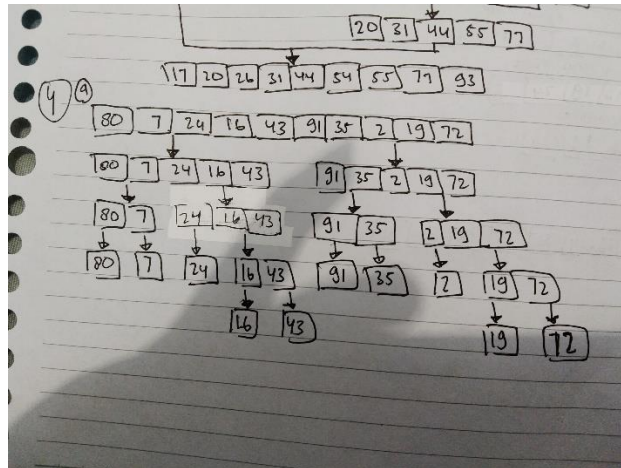
Picture 1.5

Code is speed test merge sort and quick sort

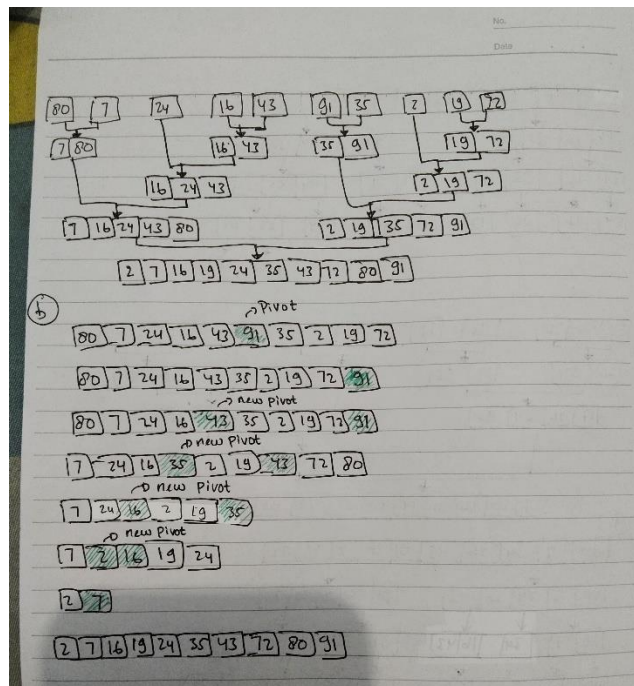


merge: 0.0297217 detik
quick: 0.0191486 detik

Picture 1.6 4.



Picture 1.7



Picture 1.8

5.

```
1  #nomer 5
2  def merge(arr, left, mid, right):
3      # Buat dua array sementara untuk menyimpan bagian kiri dan kanan
4      n1 = mid - left + 1
5      n2 = right - mid
6
7      L = [0] * n1
8      R = [0] * n2
9
10     for i in range(n1):
11         L[i] = arr[left + i]
12     for j in range(n2):
13         R[j] = arr[mid + 1 + j]
14
15     i = j = 0
16     k = left
17
18     # Merge kembali ke array utama
19     while i < n1 and j < n2:
20         if L[i].NIM ≤ R[j].NIM:
21             arr[k] = L[i]
22             i += 1
23         else:
24             arr[k] = R[j]
25             j += 1
26         k += 1
27
28     # Salin sisa elemen L (jika ada)
29     while i < n1:
30         arr[k] = L[i]
31         i += 1
32         k += 1
33
34     # Salin sisa elemen R (jika ada)
35     while j < n2:
36         arr[k] = R[j]
37         j += 1
38         k += 1
39
40     def mergeSort(arr, left, right):
41         if left < right:
42             mid = (left + right) // 2
43             mergeSort(arr, left, mid)
44             mergeSort(arr, mid + 1, right)
45             merge(arr, left, mid, right)
46
47
48
49     mergeSort(data, 0, len(data) - 1)
50     print('Setelah Sort:', data)
```

Picture 1.9

- **merge(arr, left, mid, right)** : This function merges two sorted parts (**L** and **R**) of an array into a single sorted array by comparing elements from both parts and copying them back to the main array.

- **mergeSort(arr, left, right)** : This function implements the Merge Sort algorithm recursively. The array is divided into two halves until each part contains only one element, then merged back using the **merge** function.
- Usage : The code sorts the **data** list using Merge Sort and prints the final result.

➡ Setelah Sort: [Ahmad (NIM: 2), Eka (NIM: 4), Ika (NIM: 10), Chandra (NIM: 18), Budi (NIM: 51)]

Picture 2.0

6.

```

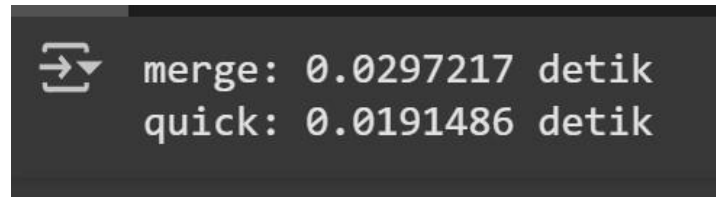
1  #nomer 6
2  import time
3  from random import shuffle
4
5  class MhsTIF:
6      def __init__(self, nama, NIM, kota, uangSaku):
7          self.nama, self.NIM, self.kota, self.uangSaku = nama, NIM, kota, uangSaku
8
9  # == Quick Sort dengan Median-of-Three ==
10 def quickSort(arr, l, r):
11     if l < r:
12         median3(arr, l, r)
13         p = partition(arr, l, r)
14         quickSort(arr, l, p - 1)
15         quickSort(arr, p + 1, r)
16
17 def median3(arr, l, r):
18     m = (l + r) // 2
19     if arr[m].NIM < arr[l].NIM: arr[m], arr[l] = arr[l], arr[m]
20     if arr[r].NIM < arr[l].NIM: arr[r], arr[l] = arr[l], arr[r]
21     if arr[r].NIM < arr[m].NIM: arr[r], arr[m] = arr[m], arr[r]
22     arr[m], arr[r] = arr[r], arr[m]
23
24 def partition(arr, l, r):
25     pivot = arr[r].NIM
26     i = l - 1
27     for j in range(l, r):
28         if arr[j].NIM ≤ pivot:
29             i += 1
30             arr[i], arr[j] = arr[j], arr[i]
31     arr[i + 1], arr[r] = arr[r], arr[i + 1]
32     return i + 1
33
34 # == Pembuatan Data Acak ==
35 def generate_data(n):
36     names = ['Ika', 'Budi', 'Ahmad', 'Chandra', 'Eka', 'Fandi', 'Deni', 'Galuh', 'Janto', 'Hasan']
37     data = [MhsTIF(names[i % len(names)], i, f'Kota {i}', 240000 + i * 1000) for i in range(n)]
38     shuffle(data)
39     return data
40
41 # == Pengujian Waktu Eksekusi ==
42 data = generate_data(6000)
43 data_quick = data[:]
44
45 t3 = time.time()
46 quickSort(data_quick, 0, len(data_quick) - 1)
47 t4 = time.time()
48
49 print('Quick Sort : %.6f detik' % (t4 - t3))

```

Picture 2.1

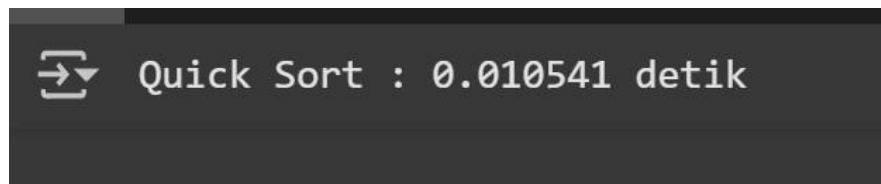
This code implements the Quick Sort algorithm with the Median-of-Three technique to sort a list of student objects based on their NIM, including random data generation and execution time measurement.

7.

A terminal window with a dark background and light gray text. It shows two lines of output: 'merge: 0.0297217 detik' and 'quick: 0.0191486 detik'. To the left of the text is a small icon consisting of a right-pointing arrow inside a square frame.

```
merge: 0.0297217 detik  
quick: 0.0191486 detik
```

Picture 2.2

A terminal window with a dark background and light gray text. It shows one line of output: 'Quick Sort : 0.010541 detik'. To the left of the text is a small icon consisting of a right-pointing arrow inside a square frame.

```
Quick Sort : 0.010541 detik
```

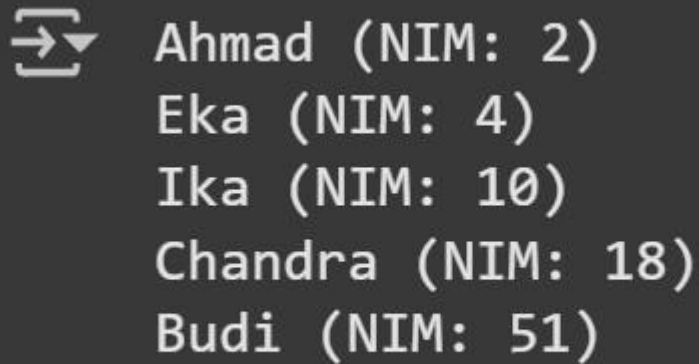
Picture 2.3

8.

```
1 #nomer 8
2 from random import shuffle
3
4 class MhsTIF:
5     def __init__(self, nama, NIM, kota, uangSaku):
6         self.nama, self.NIM, self.kota, self.uangSaku = nama, NIM, kota, uangSaku
7     def __repr__(self): return f'{self.nama} (NIM: {self.NIM})'
8
9 class Node:
10     def __init__(self, data):
11         self.data = data
12         self.next = None
13
14 def merge_sort(head):
15     if not head or not head.next:
16         return head
17     mid = get_middle(head)
18     left = merge_sort(head)
19     right = merge_sort(mid)
20     return merge(left, right)
21
22 def get_middle(head):
23     slow = fast = head
24     while fast and fast.next and fast.next.next:
25         slow = slow.next
26         fast = fast.next.next
27     next_half = slow.next
28     slow.next = None
29     return next_half
30
31 def merge(l1, l2):
32     dummy = Node(None)
33     tail = dummy
34     while l1 and l2:
35         if l1.data.NIM <= l2.data.NIM:
36             tail.next, l1 = l1, l1.next
37         else:
38             tail.next, l2 = l2, l2.next
39         tail = tail.next
40     tail.next = l1 or l2
41     return dummy.next
42
43 def to_linked_list(items):
44     head = curr = None
45     for item in items:
46         node = Node(item)
47         if not head:
48             head = curr = node
49         else:
50             curr.next = node
51             curr = node
52     return head
53
54 # === Data Tetap ===
55 data = [
56     MhsTIF('Ika', 10, 'Sukoharjo', 240000),
57     MhsTIF('Budi', 51, 'Sragen', 230000),
58     MhsTIF('Ahmad', 2, 'Surakarta', 250000),
59     MhsTIF('Chandra', 18, 'Surakarta', 235000),
60     MhsTIF('Eka', 4, 'Boyolali', 240000)
61 ]
62
63 # === Urutkan Data ===
64 head = to_linked_list(data)
65 sorted_head = merge_sort(head)
66
67 # === Tampilkan Hasil ===
68 current = sorted_head
69 while current:
70     print(current.data)
71     current = current.next
```

Picture 2.4

This code implements Merge Sort on a Linked List to sort a list of **MhsTIF** objects based on their **NIM**. It includes the **MhsTIF** class, a **Node** class for the linked list, a **merge_sort** function to divide and merge the list, and a **to_linked_list** function to convert the data into a linked list. The data is predefined, sorted using Merge Sort, and the results are printed.



```
⇒ Ahmad (NIM: 2)
   Eka (NIM: 4)
   Ika (NIM: 10)
   Chandra (NIM: 18)
   Budi (NIM: 51)
```

Picture 2.5