PRACTICUM REPORT

ALGORITHM AND DATA STRUCTURES MODUL 6 : ADVANCED SEQUENCE

Disusun Oleh :

ONIC AGUSTINO

L200234275

X

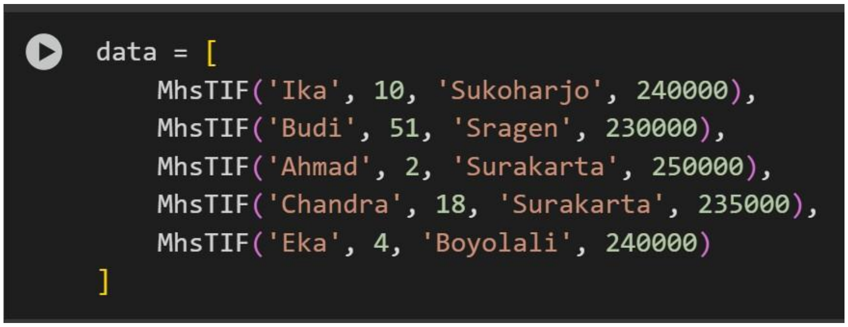
INFORMATICS ENGINEERING

FACULTY OF COMMUNICATION AND INFORMATICS

UNIVERSITAS MUHAMMADIYAH SURAKARTA

YEARS 2024/2025

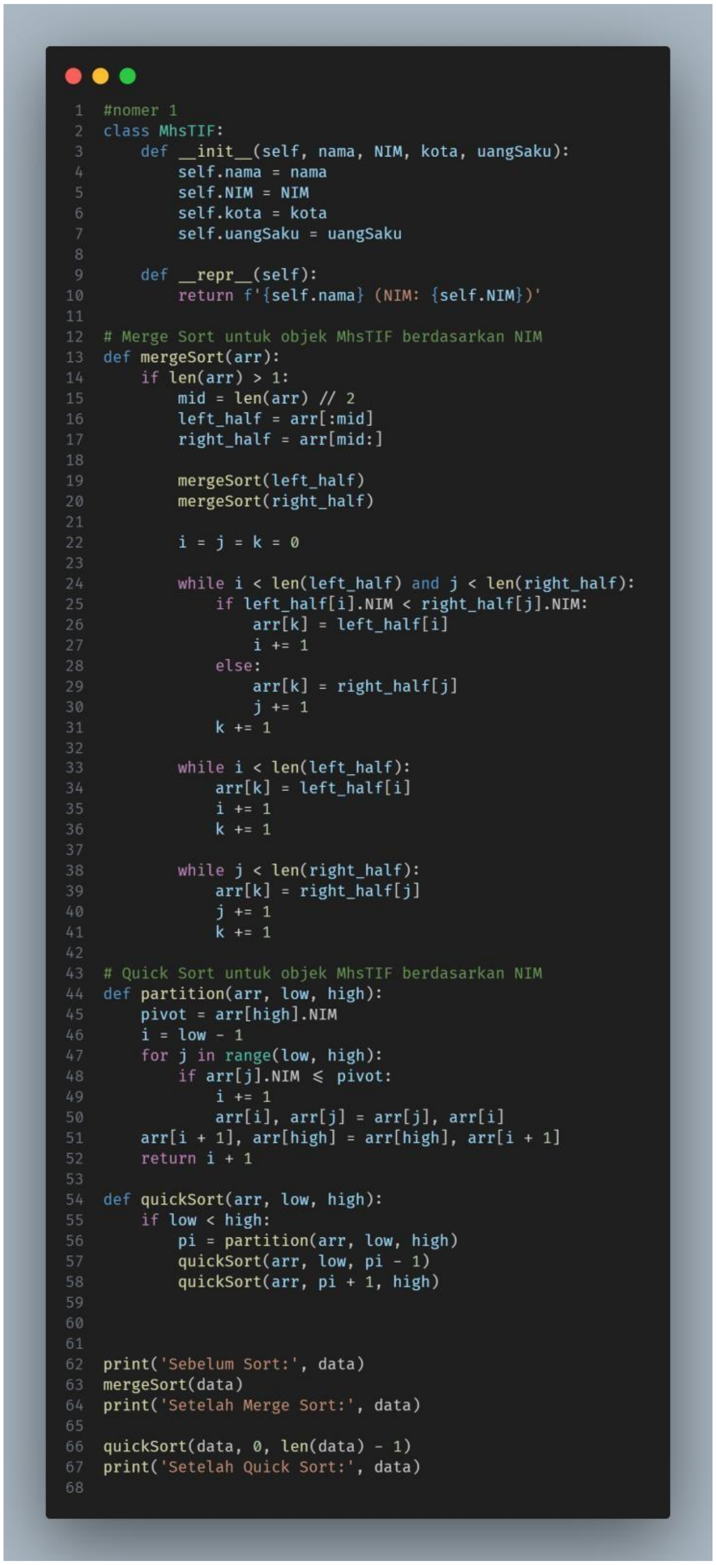
TASK:



```
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

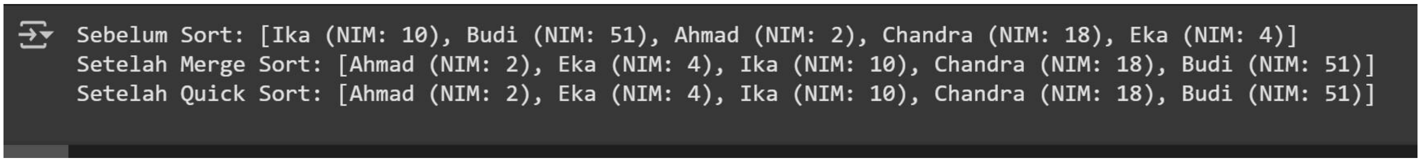
1.



```python
#nomer 1
class MhsTIF:
    def __init__(self, nama, NIM, kota, uangSaku):
        self.nama = nama
        self.NIM = NIM
        self.kota = kota
        self.uangSaku = uangSaku

    def __repr__(self):
        return f'{self.nama} (NIM: {self.NIM})'

# Merge Sort untuk objek MhsTIF berdasarkan NIM
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        mergeSort(left_half)
        mergeSort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i].NIM < right_half[j].NIM:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

# Quick Sort untuk objek MhsTIF berdasarkan NIM
def partition(arr, low, high):
    pivot = arr[high].NIM
    i = low - 1
    for j in range(low, high):
        if arr[j].NIM <= pivot:
            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)



print('Sebelum Sort:', data)
mergeSort(data)
print('Setelah Merge Sort:', data)

quickSort(data, 0, len(data) - 1)
print('Setelah Quick Sort:', data)
```

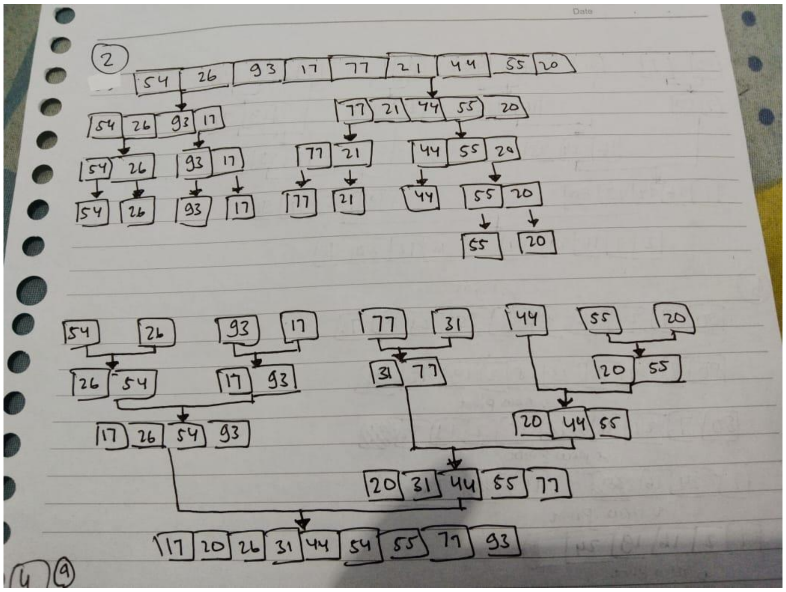
Picture 1.2

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.



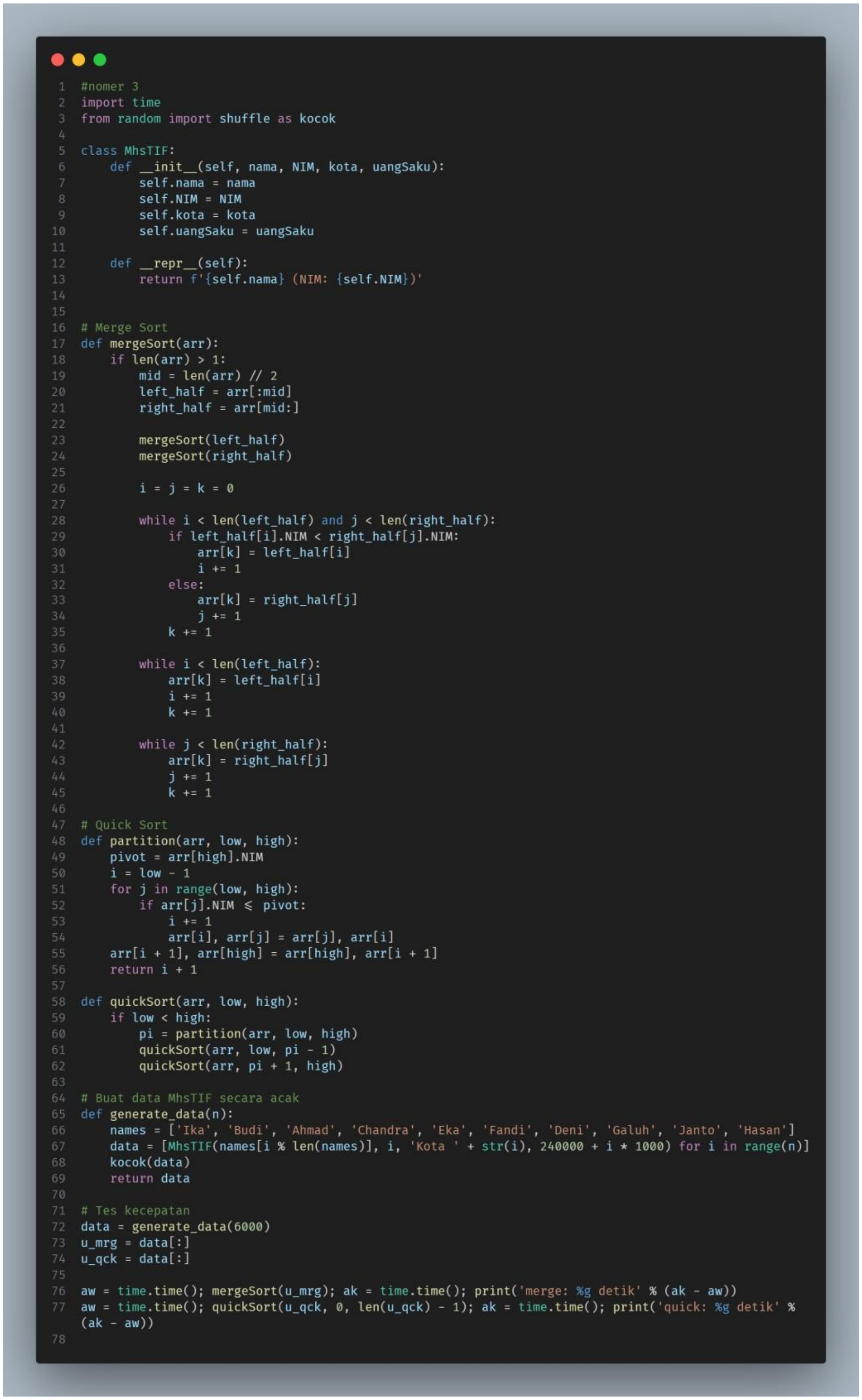
```
Sebelum Sort: [Ika (NIM: 10), Budi (NIM: 51), Ahmad (NIM: 2), Chandra (NIM: 18), Eka (NIM: 4)]
Setelah Merge Sort: [Ahmad (NIM: 2), Eka (NIM: 4), Ika (NIM: 10), Chandra (NIM: 18), Budi (NIM: 51)]
Setelah Quick Sort: [Ahmad (NIM: 2), Eka (NIM: 4), Ika (NIM: 10), Chandra (NIM: 18), Budi (NIM: 51)]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

Picture 1.3 2.



Picture 1.4

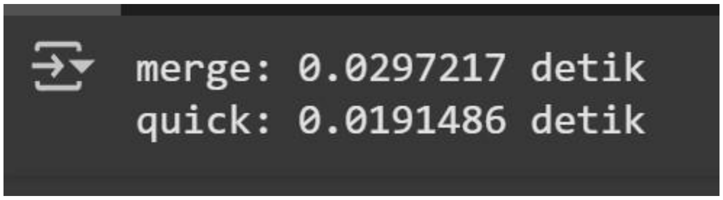
3.



```python
#nomer 3
import time
from random import shuffle as kocok

class MhsTIF:
    def __init__(self, nama, NIM, kota, uangSaku):
        self.nama = nama
        self.NIM = NIM
        self.kota = kota
        self.uangSaku = uangSaku

    def __repr__(self):
        return f'{self.nama} (NIM: {self.NIM})'


# Merge Sort
def mergeSort(arr):
    if len(arr) > 1:
        mid = len(arr) // 2
        left_half = arr[:mid]
        right_half = arr[mid:]

        mergeSort(left_half)
        mergeSort(right_half)

        i = j = k = 0

        while i < len(left_half) and j < len(right_half):
            if left_half[i].NIM < right_half[j].NIM:
                arr[k] = left_half[i]
                i += 1
            else:
                arr[k] = right_half[j]
                j += 1
            k += 1

        while i < len(left_half):
            arr[k] = left_half[i]
            i += 1
            k += 1

        while j < len(right_half):
            arr[k] = right_half[j]
            j += 1
            k += 1

# Quick Sort
def partition(arr, low, high):
    pivot = arr[high].NIM
    i = low - 1
    for j in range(low, high):
        if arr[j].NIM <= pivot:
            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)

# Buat data MhsTIF secara acak
def generate_data(n):
    names = ['Ika', 'Budi', 'Ahmad', 'Chandra', 'Eka', 'Fandi', 'Deni', 'Galuh', 'Janto', 'Hasan']
    data = [MhsTIF(names[i % len(names)], i, 'Kota ' + str(i), 240000 + i * 1000) for i in range(n)]
    kocok(data)
    return data

# Tes kecepatan
data = generate_data(6000)
u_mrg = data[:]
u_qck = data[:]

aw = time.time(); mergeSort(u_mrg); ak = time.time(); print('merge: %g detik' % (ak - aw))
aw = time.time(); quickSort(u_qck, 0, len(u_qck) - 1); ak = time.time(); print('quick: %g detik' % (ak - aw))

```

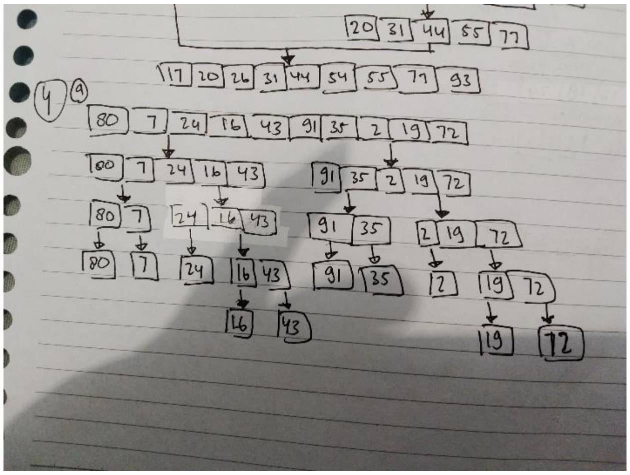
Picture 1.5

Code is speed test merge sort and quick sort

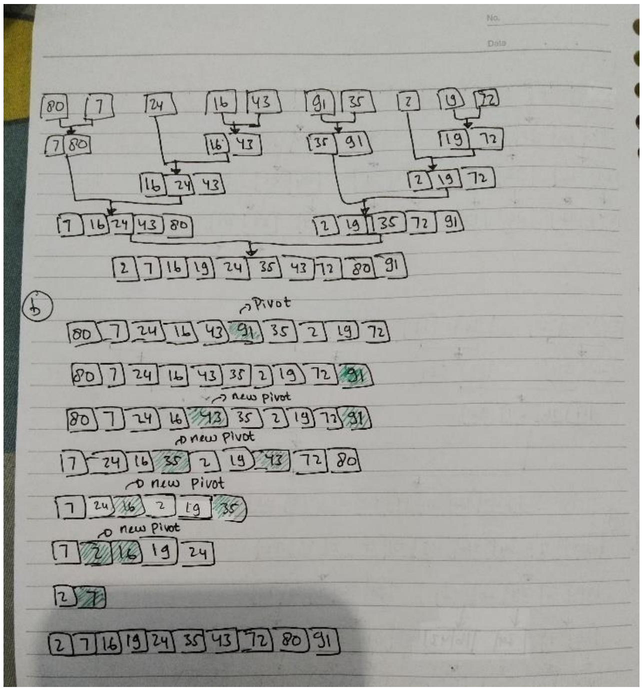


```
merge: 0.0297217 detik
quick: 0.0191486 detik
```

Picture 1.6 4.

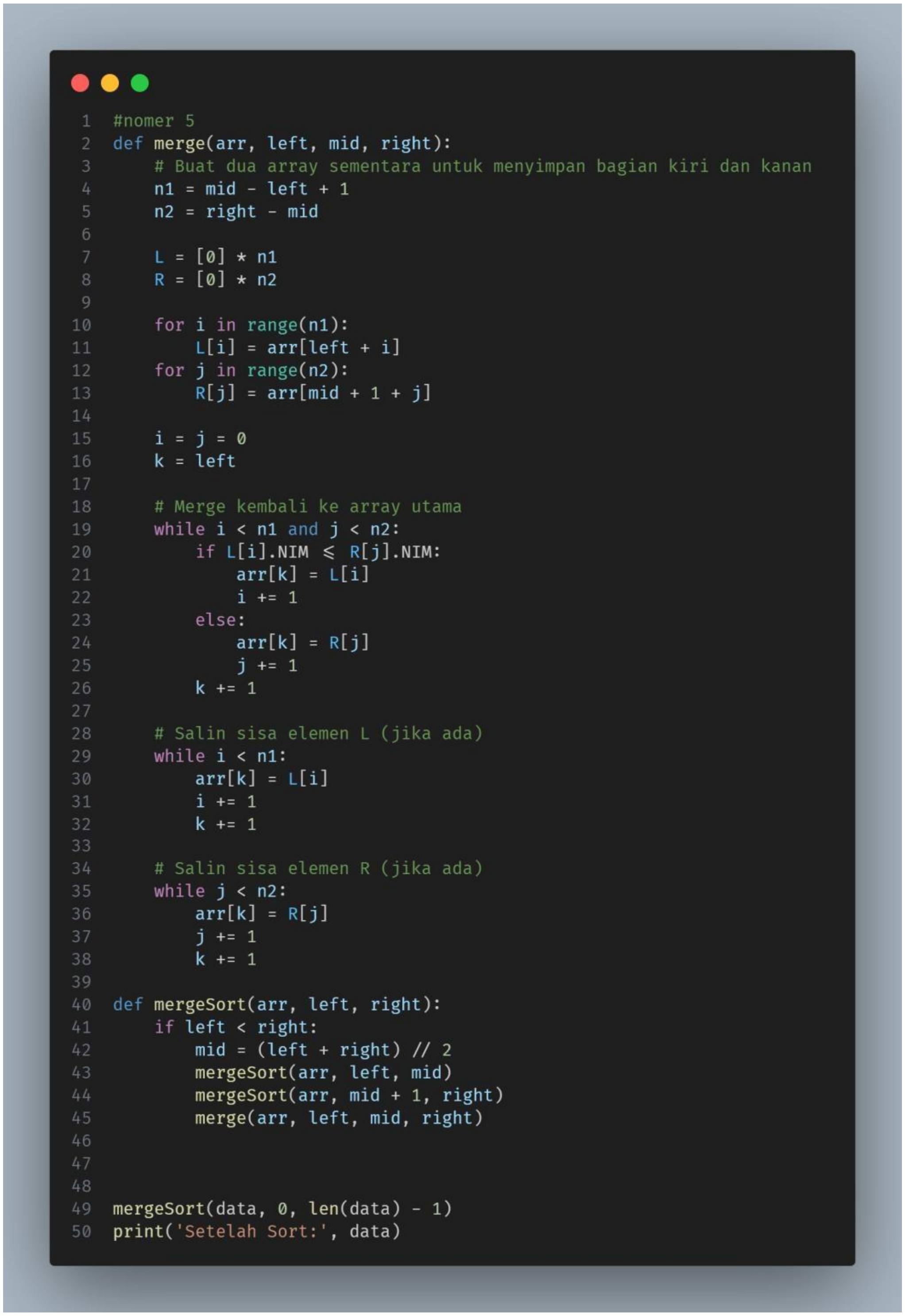


Picture 1.7



Picture 1.8

5.

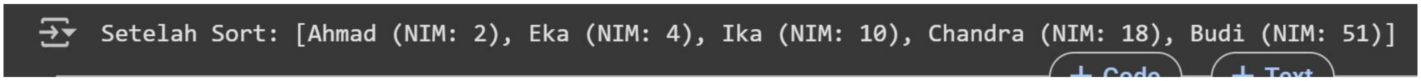


```
#nomer 5
def merge(arr, left, mid, right):
    # Buat dua array sementara untuk menyimpan bagian kiri dan kanan
    n1 = mid - left + 1
    n2 = right - mid

    L = [0] * n1
    R = [0] * n2

    for i in range(n1):
        L[i] = arr[left + i]
    for j in range(n2):
        R[j] = arr[mid + 1 + j]

    i = j = 0
    k = left

    # Merge kembali ke array utama
    while i < n1 and j < n2:
        if L[i].NIM ≤ R[j].NIM:
            arr[k] = L[i]
            i += 1
        else:
            arr[k] = R[j]
            j += 1
        k += 1

    # Salin sisa elemen L (jika ada)
    while i < n1:
        arr[k] = L[i]
        i += 1
        k += 1

    # Salin sisa elemen R (jika ada)
    while j < n2:
        arr[k] = R[j]
        j += 1
        k += 1

def mergeSort(arr, left, right):
    if left < right:
        mid = (left + right) // 2
        mergeSort(arr, left, mid)
        mergeSort(arr, mid + 1, right)
        merge(arr, left, mid, right)



mergeSort(data, 0, len(data) - 1)
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.

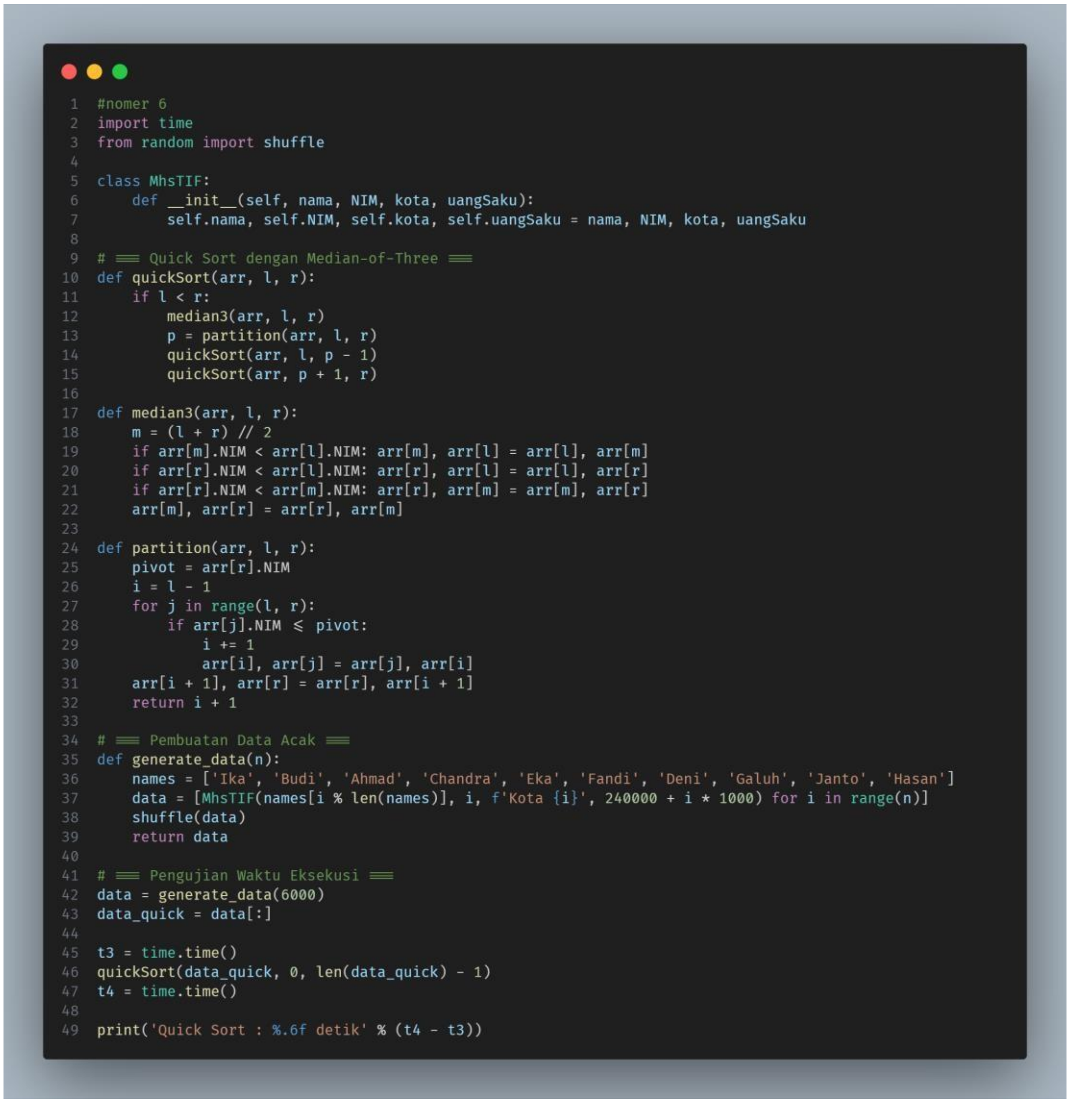
- **mergeSort(arr, left, right)** : This function implements the Merge Sort algorithm recursively. The array is divided into two halves until each part contains only one element, then merged back using the **merge** function.
- Usage : The code sorts the **data** list using Merge Sort and prints the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



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

Picture 2.0

6.

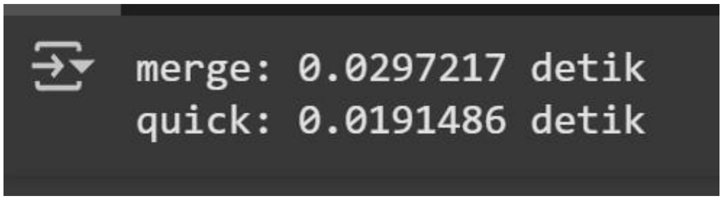


```python
#nomer 6
import time
from random import shuffle

class MhsTIF:
    def __init__(self, nama, NIM, kota, uangSaku):
        self.nama, self.NIM, self.kota, self.uangSaku = nama, NIM, kota, uangSaku

# === Quick Sort dengan Median-of-Three ===
def quickSort(arr, l, r):
    if l < r:
        median3(arr, l, r)
        p = partition(arr, l, r)
        quickSort(arr, l, p - 1)
        quickSort(arr, p + 1, r)

def median3(arr, l, r):
    m = (l + r) // 2
    if arr[m].NIM < arr[l].NIM: arr[m], arr[l] = arr[l], arr[m]
    if arr[r].NIM < arr[l].NIM: arr[r], arr[l] = arr[l], arr[r]
    if arr[r].NIM < arr[m].NIM: arr[r], arr[m] = arr[m], arr[r]
    arr[m], arr[r] = arr[r], arr[m]

def partition(arr, l, r):
    pivot = arr[r].NIM
    i = l - 1
    for j in range(l, r):
        if arr[j].NIM <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[r] = arr[r], arr[i + 1]
    return i + 1

# === Pembuatan Data Acak ===
def generate_data(n):
    names = ['Ika', 'Budi', 'Ahmad', 'Chandra', 'Eka', 'Fandi', 'Deni', 'Galuh', 'Janto', 'Hasan']
    data = [MhsTIF(names[i % len(names)], i, f'Kota {i}', 240000 + i * 1000) for i in range(n)]
    shuffle(data)
    return data

# === Pengujian Waktu Eksekusi ===
data = generate_data(6000)
data_quick = data[:]

t3 = time.time()
quickSort(data_quick, 0, len(data_quick) - 1)
t4 = time.time()

print('Quick Sort : %.6f detik' % (t4 - t3))
```

Picture 2.1

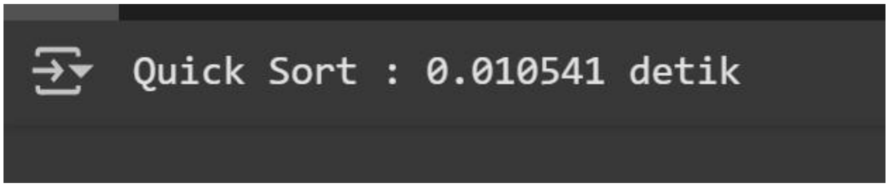
This code implements the Quick Sort algorithm with the Median-of-Three technique to sort a list of student objects based on their NIM, including random data generation and execution time measurement.

7.



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

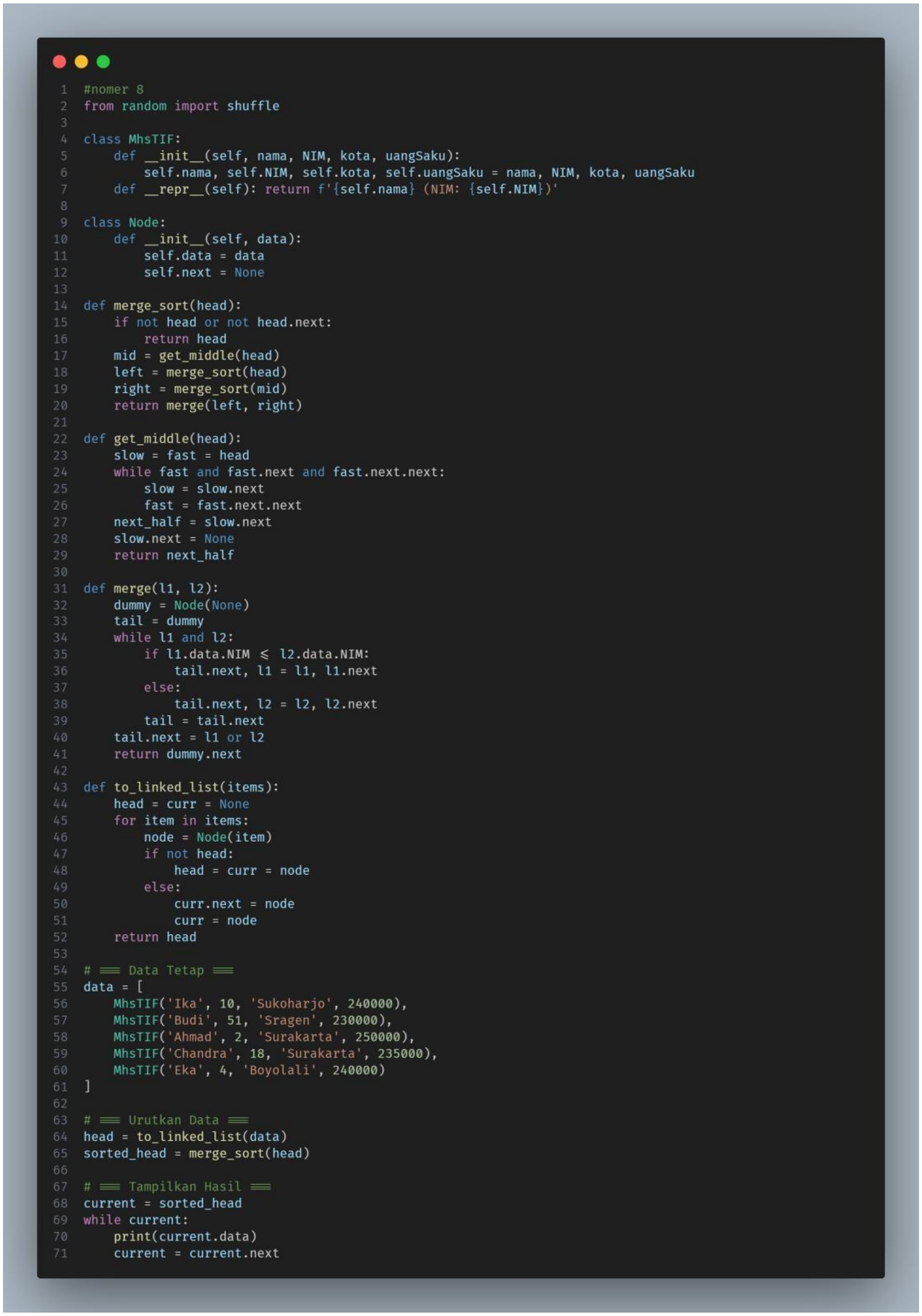
Picture 2.2



```
Quick Sort : 0.010541 detik
```

Picture 2.3

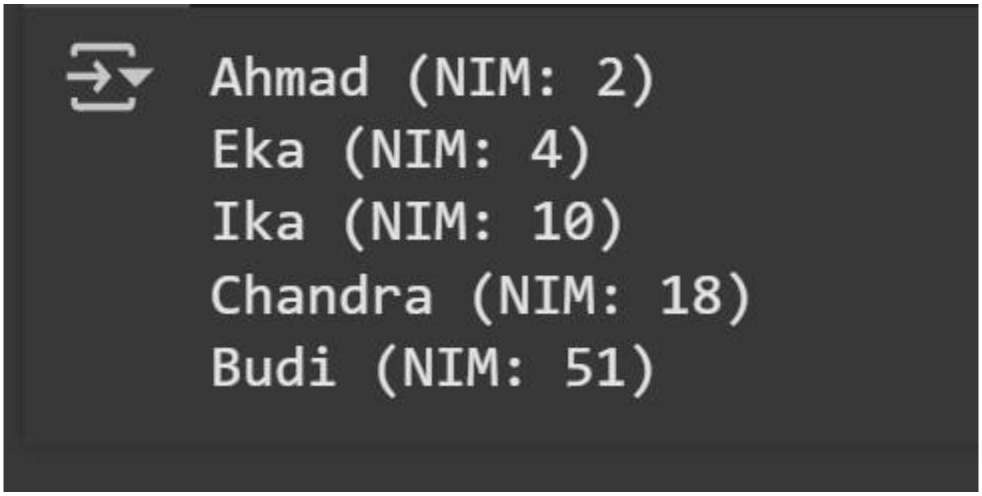
8.



```python
#nomer 8
from random import shuffle

class MhsTIF:
    def __init__(self, nama, NIM, kota, uangSaku):
        self.nama, self.NIM, self.kota, self.uangSaku = nama, NIM, kota, uangSaku
    def __repr__(self): return f'{self.nama} (NIM: {self.NIM})'

class Node:
    def __init__(self, data):
        self.data = data
        self.next = None

def merge_sort(head):
    if not head or not head.next:
        return head
    mid = get_middle(head)
    left = merge_sort(head)
    right = merge_sort(mid)
    return merge(left, right)

def get_middle(head):
    slow = fast = head
    while fast and fast.next and fast.next.next:
        slow = slow.next
        fast = fast.next.next
    next_half = slow.next
    slow.next = None
    return next_half

def merge(l1, l2):
    dummy = Node(None)
    tail = dummy
    while l1 and l2:
        if l1.data.NIM <= l2.data.NIM:
            tail.next, l1 = l1, l1.next
        else:
            tail.next, l2 = l2, l2.next
        tail = tail.next
    tail.next = l1 or l2
    return dummy.next

def to_linked_list(items):
    head = curr = None
    for item in items:
        node = Node(item)
        if not head:
            head = curr = node
        else:
            curr.next = node
            curr = node
    return head

# === Data Tetap ===
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)
]

# === Urutkan Data ===
head = to_linked_list(data)
sorted_head = merge_sort(head)

# === Tampilkan Hasil ===
current = sorted_head
while current:
    print(current.data)
    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