LAPORAN PRAKTIKUM
ALGORITMA STRUKTUR DATA
MODUL 6

Oleh:

Ammar Miftahudin Anshori (L200230270)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS KOMUNIKASI DAN INFORMATIKA

UNIVERSITAS MUHAMMADIYAH SURAKARTA

TAHUN AJARAN 2025/2026

SOAL-SOAL UNTUK MAHASISWA

1. Mengubah mergeSort dan Quick Sort agar dapat mengurutkan objek MHSTif

```python
class MhsTIF(object):
    def __init__(self, nama, nim, alamat, uang):
        self.nama = nama
        self.nim = nim
        self.alamat = alamat
        self.uang = uang

    def __str__(self):
        return f'{self.nama}, {self.nim}, {self.alamat}, {self.uang}'

```

   a. Merge Sort

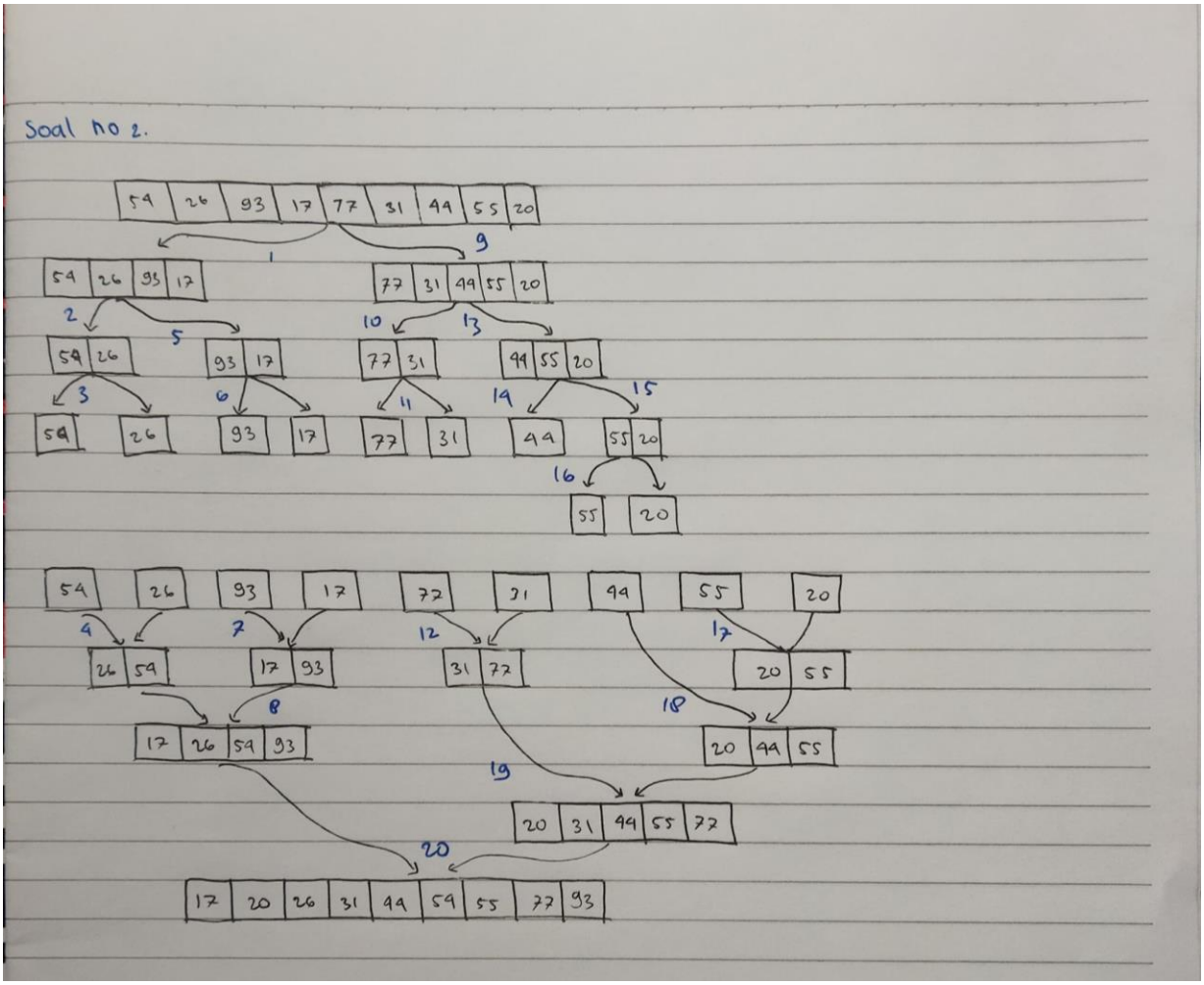
```python
def mergeSort(mergeList):
    if len(mergeList) > 1:
        tengah = len(mergeList) // 2
        separuhKiri = mergeList[:tengah]
        separuhKanan = mergeList[tengah:]

        mergeSort(separuhKiri)
        mergeSort(separuhKanan)

        i = 0
        j = 0
        k = 0
        while i < len(separuhKiri) and j < len(separuhKanan):
            if separuhKiri[i].nim < separuhKanan[j].nim:
                mergeList[k] = separuhKiri[i]
                i += 1
            else:
                mergeList[k] = separuhKanan[j]
                j += 1
            k += 1

        while i < len(separuhKiri):
            mergeList[k] = separuhKiri[i]
            i += 1
            k += 1

        while j < len(separuhKanan):
            mergeList[k] = separuhKanan[j]
            j += 1
            k += 1
    return mergeList
```

   b. Quick Sort

```python
def quickSort(list):
    quickSortBantu(list, 0, len(list) - 1)


def quickSortBantu(list, awal, akhir):
    if awal < akhir:
        titikBelah = partisi(list, awal, akhir)
        quickSortBantu(list, awal, titikBelah - 1)
        quickSortBantu(list, titikBelah + 1, akhir)


def partisi(list, awal, akhir):
    nilaiPivot = list[awal].nim
    penandaKiri = awal + 1
    penandaKanan = akhir
    selesai = False
    while not selesai:

        while penandaKiri <= penandaKanan and list[penandaKiri].nim <= nilaiPivot:
            penandaKiri += 1

        while list[penandaKanan].nim >= nilaiPivot and penandaKanan >= penandaKiri:
            penandaKanan -= 1

        if penandaKanan < penandaKiri:
            selesai = True
        else:
            temp = list[penandaKiri]
            list[penandaKiri] = list[penandaKanan]
            list[penandaKanan] = temp

    temp = list[awal]
    list[awal] = list[penandaKanan]
    list[penandaKanan] = temp

    return penandaKanan
```

Pada kode merge dan quick hanya menambahkan atau mengambil data nim yaitu integer didalam objek mahasiswa untuk proses pengurutan pada algoritma merge dan quick.

2. Menandai pengrutan merge sort



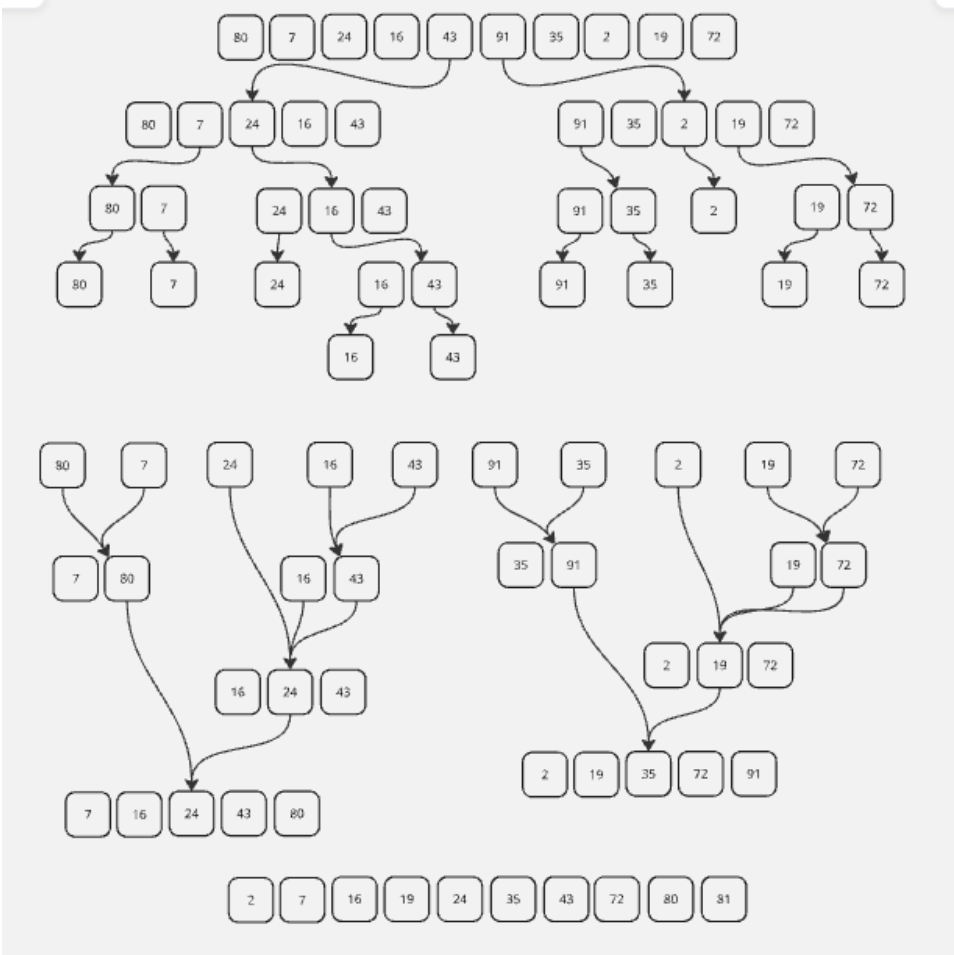
3. Menguji kecepatan dari beberapa algoritma pengurutan terutama Merge dan Quick sort
   a. Merge sort

```
aw = detak()
bubbleSort(u_bub)
ak = detak()
print('Bubble Sort: %g detik' % (ak-aw))

aw = detak()
bubbleSort(u_sel)
ak = detak()
print('Selection Sort: %g detik' % (ak-aw))

aw = detak()
bubbleSort(u_ins)
ak = detak()
print('Insertion Sort: %g detik' % (ak-aw))

aw = detak()
mergeSort(u_mrg)
ak = detak()
print('Merge Sort: %g detik' % (ak-aw))

aw = detak()
quickSort(u_qck)
ak = detak()
print('Quick Sort: %g detik' % (ak-aw))
```

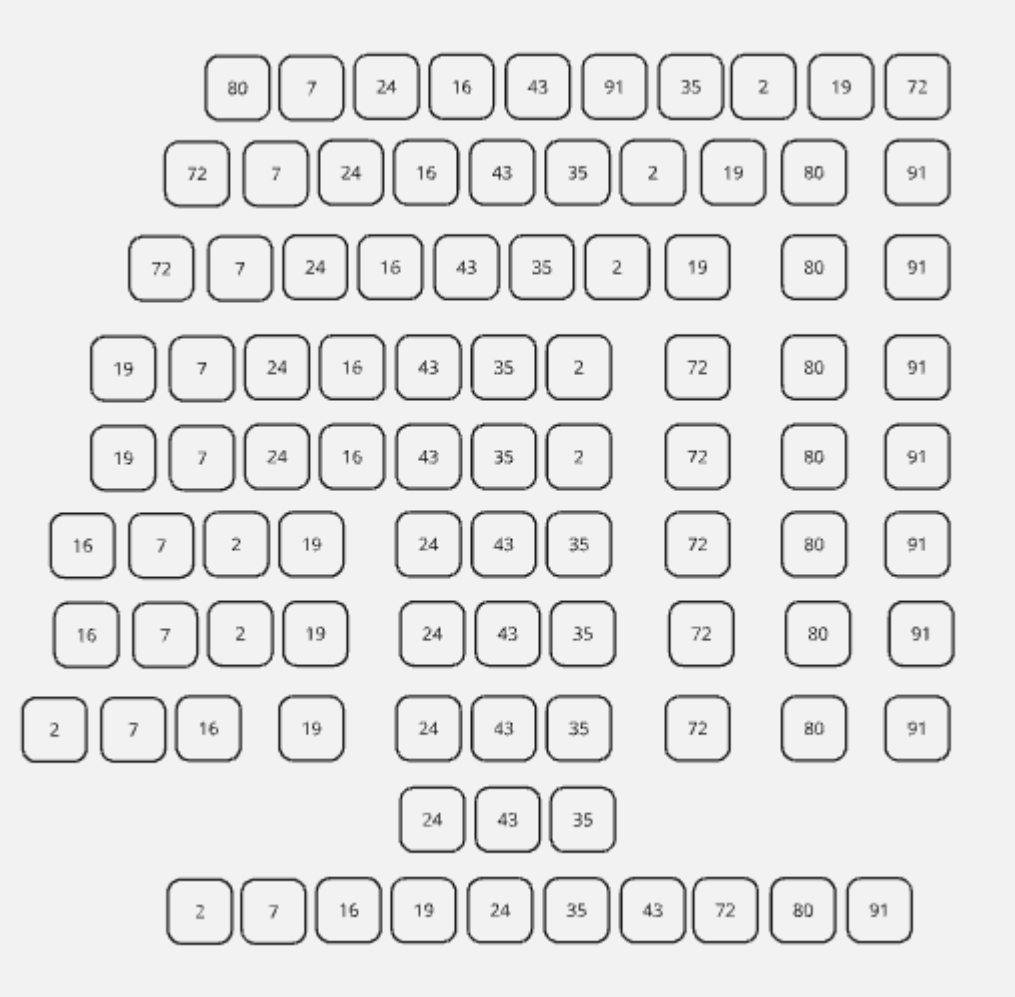
```
E:\.UMS\.SEMESTER-4\2. PRAK ASD\MODUL6>C:/
py"
Bubble Sort: 6.69972 detik
Selection Sort: 3.79258 detik
Insertion Sort: 5.31939 detik
Merge Sort: 0.0438826 detik
Quick Sort: 0.0867674 detik

E:\.UMS\.SEMESTER-4\2. PRAK ASD\MODUL6>
```

Dari hasil diatas, algoritma pengurutan yang paling efektif dengan waktu tersepat adalah quick sort dan merge sort dengan waktu dibawah 1 detik dibandingkan algoritma lain yang terpaut begitu jauh.

4. Menggambarkan pengurutan dari merge dan quick sort
   a. Merge sort


80 7 24 16 43 91 35 2 19 72
80 7 24 16 43
91 35 2 19 72
80 7
24 16 43
91 35
2
19 72
80
7
24
16 43
91
35
19
72
16
43
80 7 24 16 43 91 35 2 19 72
7 80
16 43
35 91
19 72
16 24 43
2 19 72
7 16 24 43 80
2 19 35 72 91
2 7 16 19 24 35 43 72 80 81

b. Quick sort



5. Mengubah kode Merge Sort agar efisien

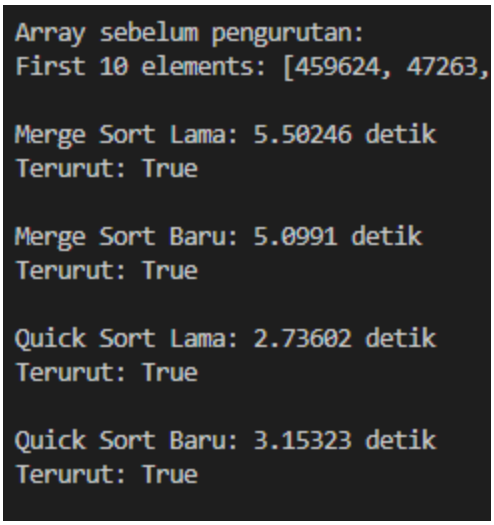
```python
def mergeSort(dataList):
    temp = [0] * len(dataList)
    mergeSortBantu(dataList, temp, 0, len(dataList) - 1)


def mergeSortBantu(dataList, temp, kiri, kanan):
    if kiri < kanan:
        tengah = (kiri + kanan) // 2
        mergeSortBantu(dataList, temp, kiri, tengah)
        mergeSortBantu(dataList, temp, tengah + 1, kanan)
        merge(dataList, temp, kiri, tengah, kanan)


def merge(dataList, temp, kiri, tengah, kanan):
    i = kiri
    j = tengah + 1
    k = kiri

    while i <= tengah and j <= kanan:
        if dataList[i] < dataList[j]:
            temp[k] = dataList[i]
            i += 1
        else:
            temp[k] = dataList[j]
            j += 1
        k += 1

    while i <= tengah:
        temp[k] = dataList[i]
        i += 1
        k += 1

    while j <= kanan:
        temp[k] = dataList[j]
        j += 1
        k += 1

    for k in range(kiri, kanan + 1):
        dataList[k] = temp[k]

```

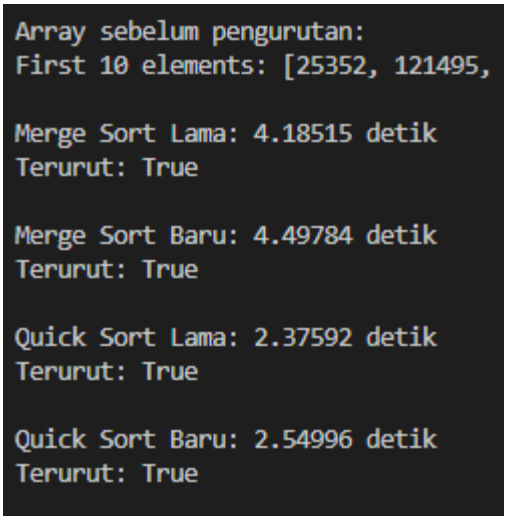
Kali ini untuk algoritma Merge Sort dibagi menjadi beberapa fungsi untuk meningkatkan efisiensi seperti quick sort.

6. Meningkatkan efisiensi quick sort menggunakan median dari tiga

```python
def median_of_three(lists, awal, akhir):
    mid = (awal + akhir) // 2
    a, b, c = lists[awal], lists[mid], lists[akhir]
    return sorted([a, b, c])[1]

def partition(lists, awal, akhir):
    pivot = median_of_three(lists, awal, akhir)
    pivot_index = lists.index(pivot)

    lists[pivot_index], lists[akhir] = lists[akhir], lists[pivot_index]

    i = awal - 1
    for j in range(awal, akhir):
        if lists[j] <= pivot:
            i += 1
            lists[i], lists[j] = lists[j], lists[i]

    lists[i + 1], lists[akhir] = lists[akhir], lists[i + 1]
    return i + 1

def quickSortBantu(lists, awal, akhir):
    if awal < akhir:
        titikBelah = partition(lists, awal, akhir)
        quickSortBantu(lists, awal, titikBelah - 1)
        quickSortBantu(lists, titikBelah + 1, akhir)

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

```

7. Membandingkan hasil waktu dari kode lama dan kode baru





Dari hasil tersebut dapat dilihat bahwa efisiensi tidak selalu berpengaruh pada kecepatan sorting yang dihasilkan, kadang bisa lebih cepar kdang juga bisa lebih lambat tergantung penggunaan dengan kasus tertentu misal dengan panjang array yang berbeda-beda.

8. Membuat merge sort untuk linked list

```python
class Node(object):
    def __init__(self, data, next = None, prev = None):
        self.data = data
        self.next = next
        self.prev = prev

def mergeSortLinkedList(head):
    if head is None or head.next is None:
        return head

    separuh_kiri, separuh_kanan = pisah_list(head)

    kiri_sorted = mergeSortLinkedList(separuh_kiri)
    kanan_sorted = mergeSortLinkedList(separuh_kanan)

    return merge_lists(kiri_sorted, kanan_sorted)

def pisah_list(head):
    if head is None or head.next is None:
        return head, None

    a = head
    b = head.next

    while b and b.next:
        a = a.next
        b = b.next.next

    separuh_kanan = a.next
    a.next = None

    return head, separuh_kanan

def merge_lists(kiri, kanan):
    object_percobaan = Node(0)
    tail = object_percobaan

    while kiri and kanan:
        if kiri.data <= kanan.data:
            tail.next = kiri
            kiri = kiri.next
        else:
            tail.next = kanan
            kanan = kanan.next
        tail = tail.next

    if kiri:
        tail.next = kiri
    else:
        tail.next = kanan

    return object_percobaan.next

d1 = Node(56)
d2 = Node(12)
d3 = Node(5)
d4 = Node(25)
d5 = Node(37)

d1.next = d2
d2.next = d3
d3.next = d4
d4.next = d5

print(f'\nSebelum diurutkan: {(d1.data, d1.next.data, d1.next.next.data, d1.next.next.next.data, d1.next.next.next.next.data)}\n')

mergeSortLinkedList(d1)

print(f'Setelah diurutkan: {(d3.data, d3.next.data, d3.next.next.data, d3.next.next.next.data, d3.next.next.next.next.data)}')

```