PRAKTIKUM 13-14
ALGORITMA PENGURUTAN (QUICK DAN MERGE)

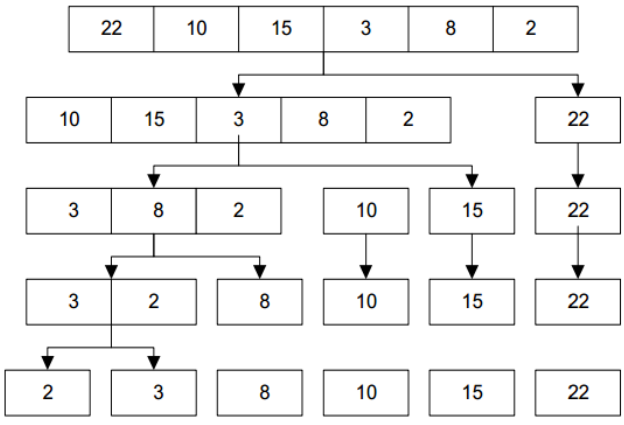
A. TUJUAN PEMBELAJARAN

1. Memahami mengenai algoritma pengurutan *quick sort* dan *merge sort.*
2. Mampu mengimplementasikan algoritma pengurutan *quick sort* dan *merge sort* secara *ascending* dan *descending.*

B. DASAR TEORI

1. Algoritma *Quick Sort*

Metode *quick sort* dikembangkan oleh C.A.R Hoare.Secara garis besar metode ini dijelaskan sebagai berikut. Misalnya kita ingin mengurutkan data A yang mempunyai N elemen. Kita pilih sembarang elemen dari data tersebut, bisanya elemen pertama, misalnya X. kemudian semua elemen tersebut disusun dengan menempatkan X pada posisi J sedemikian rupa sehingga elemen ke 1 sampai ke J 1 mempunyai nilai lebih kecil dari X dan elemen J+1sampai ke N mempunyai nilai lebih besar dari X. Sampai saat ini kita sudah mempunyai dua sub data (kiri dan kanan). Langkah berikutnya diulang untuk setiap sub data**.**



Gambar 1. Pengurutan data menggunakan algoritma Quick Sort

2. Algoritma *Merge Sort*

Algoritma *merge* ini disesuaikan untuk mesin drive tape. Penggunaannya dalam akses memori acak besar yang terkait telah menurun, karena banyak aplikasi algoritma *merge* yang mempunyai alternatif lebih cepat ketika kamu memiliki akses memori acak yang menjaga semua data mu.Hal ini disebabkan algoritma ini membutuhkan setidaknya ruang atau memori dua kali lebih besar karena dilakukan secara rekursif dan memakai dua tabel.

Algoritma *merge sort* membagi tabel menjadi dua tabel yang sama besar. Masing-masing tabel diurutkan secara rekursif, dan kemudian digabungkan kembali untuk membentuk tabel yang terurut.Implementasi dasar dari algoritma *merge sort* memakai tiga buah tabel, dua untuk menyimpan elemen dari tabel yang telah di bagi dua dan satu untuk menyimpan elemen yang telah terurut.Namun algoritma ini dapat juga dilakukan langsung pada dua tabel, sehingga menghemat ruang atau memori yang dibutuhkan.

Algoritma *Merge* umumnya memiliki satu set *pointer* $p_{0..n}$ yang menunjuk suatu posisi di dalam satu set daftar $L_{0..n}$ .Pada awalnya mereka menunjuk item yang pertama pada setiap daftar. Algoritmanya sebagai berikut:

Selama $p_{0..n}$ masih menunjuk data yang di dalam sebagai pengganti pada akhirnya:

- Melakukan sesuatu dengan data item yang menunjuk daftar mereka masing-masing.
- Menemukan *pointers points* untuk item dengan kunci yang paling rendah; membantu salah satu pointer untuk item yang berikutnya dalam daftar.

  CONTOH :

  Contoh penerapan atas sebuah larik sebagai data sumber yang akan diurutkan ***{3, 9, 4, 1, 5, 2}*** adalah sebagai berikut:
- Larik tersebut dibagi menjadi dua bagian, ***{3, 9, 4}*** dan ***{1, 5, 2}***
- Kedua larik kemudian diurutkan secara terpisah sehingga menjadi ***{3, 4, 9}*** dan ***{1, 2, 5}***
- Sebuah larik baru dibentuk yang sebagai penggabungan dari kedua larik tersebut ***{1}***, sementara nilai-nilai dalam masing larik ***{3, 4, 9}*** dan ***{2, 5}*** (nilai 1 dalam elemen larik ke dua telah dipindahkan ke larik baru)

- langkah berikutnya adalah penggabungan dari masing-masing larik ke dalam larik baru yang dibuat sebelumnya.

  a. ***{1, 2}*** **<->** ***{3, 4, 9}*** dan ***{5}***
  b. ***{1, 2, 3}*** **<->** ***{4, 9}*** dan ***{5}***
  c. ***{1, 2, 3, 4}*** **<->** ***{9}*** dan ***{5}***
  d. ***{1, 2, 3, 4, 5}*** **<->** ***{9}*** dan ***{null}***
  e. ***{1, 2, 3, 4, 5, 9}*** **<->** ***{null}*** dan ***{null}***

C. TUGAS PENDAHULUAN

Jawablah pertanyaan berikut ini :

1. Tuliskan algoritma pengurutan *quick sort* secara *ascending.*
2. Tuliskan algoritma pengurutan *merge sort* secara *ascending.*

D. PERCOBAAN

Percobaan 1 : *Quick sort* secara *ascending.*

```
public class Quick {
    public static <T extends Comparable <?super T>>  int Partition
(T[]arr,int p,int r){

        int i,j;
        T pivot, temp;

        pivot=arr[p]; //pivot pada index 0
        i=p;
        j=r;
        while (i<=j){
            while(pivot.compareTo(arr[j])<0)
                j--;
            while(pivot.compareTo(arr[i])>0)
                i++;
            if(i<j){
                temp=arr[i];
                arr[i]=arr[j];
                arr[j]=temp;
                j--;
                i++;
            }
            else
                return j;
```

```
        }
        return j;
    }

    private static <T extends Comparable<?super T>> void
Quicksort(T[]arr,int p,int r){
        int q;
        if(p<r) {
            q=Partition(arr, p, r);
            Quicksort(arr, p, q);
            Quicksort(arr, q+1, r);
        }
    }

     public static <T> void tampil(T data[]) {
        for (T objek : data) {
            System.out.print(objek + " ");
        }
        System.out.println("");
    }

    public static void main(String[] args) {
        Integer data[] = new Integer[10];
        for(int a=0 ; a<data.length ; a++)
        {
            data[a]= (int)(Math.random()*13+1);
        }
        tampil(data);
        long awal = System.currentTimeMillis();
        Quicksort(data,0,data.length-1);
        long sisaWaktu = System.currentTimeMillis() - awal;
        tampil(data);
        System.out.println("Waktu " + sisaWaktu);
    }
}
```

Percobaan 2 : *Merge sort* secara *ascending*

```
public class Merge {
    public static <T extends Comparable<? super T>> void merge(T[] arr,
int left, int median, int right) {
        Object[] temp=new Object[arr.length];
        int kiri1, kanan1, kiri2, kanan2, i, j;
        kiri1=left;
        kanan1=median;
        kiri2=median+1;
        kanan2=right;
        i=left;

        while ((kiri1 <= kanan1) && (kiri2 <= kanan2)) {
            if (arr[kiri1].compareTo(arr[kiri2]) <= 0) {
               temp[i]=arr[kiri1];
               kiri1++;
```

```
            } else {
                temp[i]=arr[kiri2];
                kiri2++;
            }
            i++;
        }
        while(kiri1<=kanan1){
            temp[i]=arr[kiri1];
            kiri1++;
            i++;
        }
        while(kiri2<=kanan2){
            temp[i]=arr[kiri2];
            kiri2++;
            i++;
        }
        j=left;
        while(j<=right){
            arr[j]=(T)temp[j];
            j++;
        }
    }

    public static <T extends Comparable<? super T>> void mergeSort(T[]
arr, int l, int r) {
        int med;
        if(l<r){
            med=(l+r)/2;
            mergeSort(arr,l,med);
            mergeSort(arr,med+1,r);
            merge(arr,l,med,r);
        }
    }

    public static <T> void tampil(T data[]) {
        for (T objek : data) {
            System.out.print(objek + " ");
        }
        System.out.println("");
    }

    public static void main(String[] args) {
        Integer data[] = new Integer[10];
        for (int i = 0; i < data.length; i++) {
            data[i] = (int) (Math.random() * 30 + 1);
        }
        tampil(data);
        long awal = System.currentTimeMillis();
        mergeSort(data, 0, data.length - 1);
        long akhir = System.currentTimeMillis() - awal;
        tampil(data);
        System.out.println("\nwaktu : " + akhir);
    }
}
```

E. LATIHAN

1. Dari percobaan 1 tambahkan method untuk melakukan pengurutan *quick sort* secara *descending*.
2. Dari percobaan 2 tambahkan method untuk melakukan pengurutan *merge sort* dengan *flag* secara *descending*.
3. Buatlah class `Mahasiswa` dengan variable `nrp` dan `nama` yang memiliki tipe `String`! Class `Mahasiswa` mengimplementasikan interface `Comparable`, selanjutnya implementasikan fungsi abstract `compareTo()`, untuk membandingkan dua objek mahasiswa berdasarkan `nrp`.

```
public class Mahasiswa implements Comparable <Mahasiswa> {
    private String nrp ;
    private String nama ;
    @Override
    public int compareTo(Mahasiswa o) {…}

    @Override
    public String toString() {…}
}
```

Lakukan pengujian fungsi `QuickSort()` lagi, sebelumnya tambahkan pada fungsi main() seperti di bawah ini !

```
public class Latihan {
   public static <T extends Comparable <?super T>>  int Partition
(T[]arr,int p,int r){
      …….
   }
   private static <T extends Comparable<?super T>> void
Quicksort(T[]arr,int p,int r){
      …….
   }
   public static <T> void tampil(T data[]) {
      …….
   }
   public static void main(String[] args) {
        Mahasiswa arr8[] = {new Mahasiswa("02", "Budi"),
                            new Mahasiswa("01", "Andi"),
                            new Mahasiswa("04", "Udin"),
                            new Mahasiswa("03", "Candra")};
        long awal = System.currentTimeMillis();
        bubbleSort (arr8);
```

```
        tampil(arr8);
        long sisaWaktu = System.currentTimeMillis() - awal;
        System.out.println("Waktu " + sisaWaktu);
    }
}
```

4. Pada soal 3 lakukan untuk algoritma *merge sort*.
5. Buatlah menu untuk memilih jenis algoritma pengurutan dan menu ascending atau descending seperti di bawah ini

```
ALGORITMA SORTING
1. Insertion
2. Selection
3. Bubble
4. Bubble dengan Flag
5. Shell
6. Quick
7. Merge
Pilihan : _1
1. Ascending
2. Descending
Pilihan : _1
Waktu : 4
```

   Gunakan data acak dari array of Integer dengan ukuran 100.
6. Lakukan percobaan untuk melihat waktu running untuk setiap algoritma sorting dengan data acak yang sama seperti table di bawah ini dan gambarkan dalam bentuk grafik (boleh menggunakan Excel atau Matlab).

   Tabel 1. Analisa Perbandingan metode pengurutan berdasarkan waktu running

Jumlah data	**Waktu Proses**						
	Insertion	**Selection**	**Bubble**	**Bubble Flag**	**Shell**	**Quick**	**Merge**
50000							
100000							
150000							
200000							
250000							
300000							
350000							
400000							
450000							
500000							

F. LAPORAN RESMI

Kerjakan hasil percobaan(D) dan latihan(E) di atas dan tambahkan analisa.