PRAKTIKUM 13
ALGORITMA PENGURUTAN (QUICK SORT)

A. TUJUAN PEMBELAJARAN

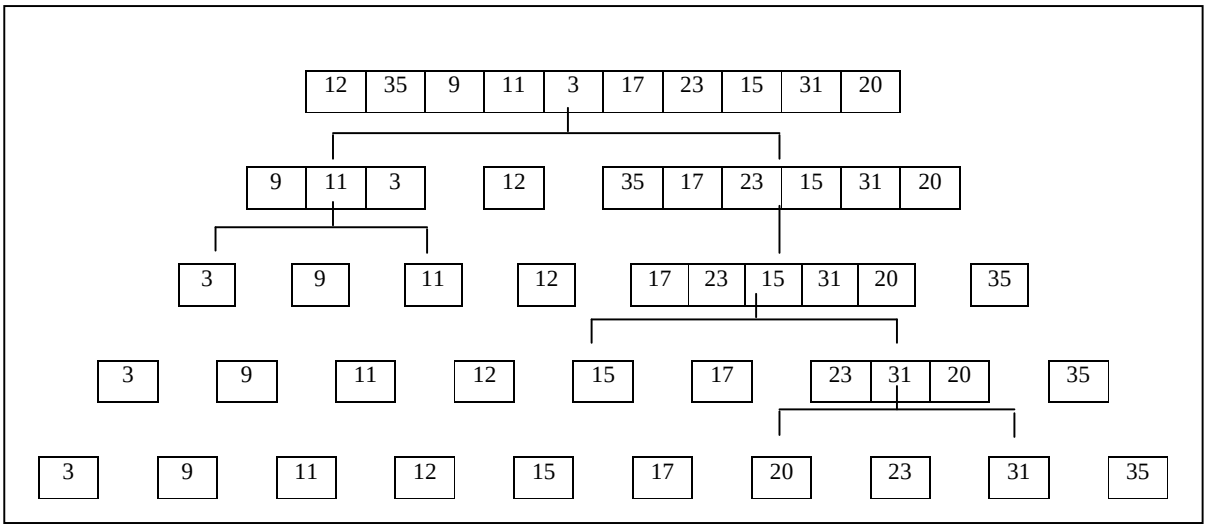
1. Memahami step by step algoritma pengurutan *quick sort.*
2. Mampu mengimplementasikan algoritma pengurutan quick sort dengan berbagai macam parameter berupa tipe data primitif atau tipe Generic.
3. Mampu mengimplementasikan algoritma pengurutan quick sort secara ascending dan descending.

B. DASAR TEORI

Algoritma *Quick Sort*

Metode Quick sering disebut juga metode partisi (*partition exchange sort*). Metode ini diperkenalkan pertama kali oleh C.A.R. Hoare pada tahun 1962. Untuk mempertinggi efektifitas dari metode ini, digunakan teknik menukarkan dua elemen dengan jarak yang cukup besar.

Proses penukaran dengan metode quick dapat dijelaskan sebagai berikut.: mula-mula dipilih data tertentu yang disebut pivot, misalnya x. Pivot dipilih untuk mengatur data di sebelah kiri agar lebih kecil daripada pivot dan data di sebelah kanan agar lebih besar daripada pivot. Pivot ini diletakkan pada posisi ke j sedemikian sehingga data antara 1 sampai dengan j-1 lebih kecil daripada x. Sedangkan data pada posisi ke j+1 sampai N lebih besar daripada x. Caranya dengan menukarkan data diantara posisi 1 sampai dengan j-1 yang lebih besar daripada x dengan data diantara posisi j+1 sampai dengan N yang lebih kecil daripada x. Ilustrasi dari metode quick dapat dilihat pada Gambar 6.1



Gambar 1 Ilustrasi Metode Quick Sort

Gambar 1 diatas menunjukkan pembagian data menjadi sub-subbagian. Pivot dipilih dari data pertama tiap bagian maupun sub bagian, tetapi sebenarnya kita bisa memilih sembarang data sebagai pivot. Dari ilustrasi diatas bisa kita lihat bahwa metode Quick Sort ini bisa kita implementasikan menggunakan dua cara, yaitu dengan cara non rekursif dan rekursif. Pada kedua cara diatas, persoalan utama yang perlu kita perhatikan adalah bagaimana kita meletakkan suatu data pada posisinya yang tepat sehingga memenuhi ketentuan diatas dan bagaimana menyimpan batas-batas subbagian. Dengan cara seperti yang diperlihatkan pada Gambar 6.1, kita hanya menggerakkan data pertama sampai di suatu tempat yang sesuai. Dalam hal ini kita hanya bergerak dari satu arah saja. Untuk mempercepat penempatan suatu data, kita bisa bergerak dari dua arah, kiri dan kanan. Caranya adalah sebagai berikut : misalnya kia mempunyai 10 data (N=9) :

12	35	9	11	3	17	23	15	31	20
i=0									j=9

Pertama kali ditentukan i=0 (untuk bergerak dari kiri ke kanan), dan j=N (untuk bergerak dari kanan ke kiri). Proses akan dihentikan jika nilai i lebih besar atau sama dengan j. Sebagai contoh, kita akan menempatkan elemen pertama, 12 pada posisinya yang tepat

dengan bergerak dari dua arah, dari kiri ke kanan dan dari kanan ke kiri secara bergantian. Dimulai dari data terakhir bergerak dari kanan ke kiri (j dikurangi 1), dilakukan pembandingan data sampai ditemukan data yang nilainya lebih kecil dari 12 yaitu 3 dan kedua elemen data ini kita tukarkan sehingga diperoleh

3	35	9	11	12	17	23	15	31	20
i=0				j=4					

Setelah itu bergerak dari kiri ke kanan dimulai dari data 3 (i ditambah 1), dilakukan pembandingan pada setiap data yang dilalui dengan 12, sampai ditemukan data yang nilainya lebih besar dari 12 yaitu 35. Kedua data kita tukarkan sehingga diperoleh

3	12	9	11	35	17	23	15	31	20
	i=1			j=4					

Berikutnya bergerak dari kanan ke kiri dimulai dari 11. Dan ternyata data 11 lebih kecil dari 12, kedua data ini ditukarkan sehingga diperoleh

3	11	9	12	35	17	23	15	31	20
	i=1		j=3						

Kemudian dimulai dari 9 bergerak dari kiri ke kanan. Pada langkah ini ternyata tidak ditemukan data yang lebih besar dari 12 sampai nilai i=j. Hal ini berarti proses penempata data yang bernilai 12 telah selesai, sehingga semua data yang lebih kecil dari 12 berada di sebelah kiri dan data yang lebih besar dari 12 berada di sebelah kanan seperti terlihat di bawah ini

3	11	9

12

35	17	23	15	31	20

C. TUGAS PENDAHULUAN

Jelaskan algoritma pengurutan *quick sort* secara *ascending* dengan data 5 6 3 1 2

D. PERCOBAAN

Percobaan 1 : *Quick sort* secara *ascending* dengan data int.

```
 public class QuickDemo {

    private static int partition(int[] A, int p, int r) {
        int pivot, i, j;

        pivot = A[p];
        i = p - 1;
        j = r + 1;
        for (;;) {

            do {
                i++;
            } while (A[i] < pivot);

            do {
                j--;
            } while (A[j] > pivot);

            if (i < j) {
                int temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            } else {
                return j;
            }
        }
    }

    public static void quickSort(int[] A, int p, int r) {
        int q;
        if (p < r) {
            q = partition(A,p,r);
            quickSort(A,p, q);
            quickSort(A,q + 1, r);
        }
    }

    public static void tampil(int data[]){
         for(int i=0;i<data.length;i++)
             System.out.print(data[i]+" ");
         System.out.println();
    }
}

public class MainQuick {
    public static void main(String[] args) {
        int A[] = {10,6,8,3,1};
        QuickDemo.tampil(A);
        QuickDemo.quickSort(A,0,A.length-1);
        QuickDemo.tampil(A);
```

```
    }
}
```

Percobaan 2 : *Quick sort* secara *ascending* dengan data double.

```
public class QuickDemo {
    private static int partition(double[] A, int p, int r) {
        int  i, j;
        double pivot;

        pivot = A[p];
        i = p - 1;
        j = r + 1;
        for (;;) {

            do {
                i++;
            } while (A[i] < pivot);

            do {
                j--;
            } while (A[j] > pivot);

            if (i < j) {
                double temp = A[i];
                A[i] = A[j];
                A[j] = temp;
            } else {
                return j;
            }
        }
    }

    public static void quickSort(double[] A, int p, int r) {
        int q;
        if (p < r) {
            q = partition(A,p,r);
            quickSort(A,p, q);
            quickSort(A,q + 1, r);
        }
    }

    public static void tampil(double data[]){
         for(int i=0;i<data.length;i++)
             System.out.print(data[i]+" ");
         System.out.println();
    }
}

public class MainQuick2 {
    public static void main(String[] args) {
        double A[] = {10.3,6.2,8.4,3.6,1.1};
        QuickDemo.tampil(A);
```

```
        QuickDemo.quickSort(A,0,A.length-1);
        QuickDemo.tampil(A);
    }
}
```

E. LATIHAN

1. Buatlah program sorting Quick dengan parameter array Integer (class Wrapper) !

```
public static void quickSort(Integer[] A){…}
```

2. Buatlah program sorting Quick dengan parameter array Double (class Wrapper) !

```
public static void quickSort(Double[] A){…}
```

3. Buatlah fungsi tampil() untuk menampilkan data.

```
public static<T> void tampil(T data[]){ }
```

4. Lakukan pengujian fungsi quickSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo1 {
    public static void main(String[] args) {
        //Data Integer
        Integer arr3[] = {1,5,6,2,8,9};
        QuickDemo.quickSort(arr3, 0,arr3.length-1);
        QuickDemo.tampil(arr3);

        //data Double
        Double arr4[] = {1.3,5.2,6.6,2.7,8.8,9.1};
        QuickDemo.quickSort(arr4,0,arr4.length-1);
        QuickDemo.tampil(arr4);
    }
}
```

5. Buatlah program sorting Quick dengan parameter array Number !

```
public static<T extends Number> void quickSort(T[] A){…}
```

6. Lakukan pengujian fungsi quickSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo2 {
    public static void main(String[] args) {
        Float arr5[] = {1.3f,5.2f,6.6f,2.7f,8.8f,9.1f};
        QuickDemo.quickSort(arr5, 0,arr5.length-1);
        QuickDemo.tampil(arr5);
```

```
        Byte arr6[] = {6,7,11,1,3,2};
        QuickDemo.quickSort(arr6, 0,arr6.length-1);
        QuickDemo.tampil(arr6);
    }
}
```

7. Buatlah program sorting Quick dengan parameter array yang memenuhi T extends Comparable !

```
public static <T extends Comparable> void quickSort2(T[] arr) {
}
```

8. Lakukan pengujian fungsi quickSort(), dengan membuat fungsi main() sebagai berikut :

```
public class Demo3 {
    public static void main(String[] args) {
        //data String
        String arr7[]=
{"jeruk","anggur","belimbing","jambu","kelengkeng"};
        QuickDemo.quickSort2(arr7, 0,arr7.length-1);
        QuickDemo.tampil(arr7);
    }
}
```

9. 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() {…}
}
```

10. Lakukan pengujian fungsi quickSort() lagi, sebelumnya tambahkan pada fungsi main() seperti di bawah ini !

```
public class Demo4 {
    public static void main(String[] args) {
```

```
        Mahasiswa arr8[] = {new Mahasiswa("02", "Budi"), new
Mahasiswa("01", "Andi"), new Mahasiswa("04", "Udin"), new
Mahasiswa("03", "Candra")};
        QuickDemo.quickSort2(arr8, 0,arr8.length-1);
        QuickDemo.tampil(arr8);
    }
}
```

F. LAPORAN RESMI

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