PRAKTIKUM
ALGORITMA PENGURUTAN (SHELL SORT)

A. TUJUAN PEMBELAJARAN

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

B. DASAR TEORI

Shell Sort

Metode ini disebut juga dengan metode pertambahan menurun (diminishing increment). Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959, sehingga sering disebut dengan Metode Shell Sort. Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu, kemudian dilakukan penukaran bila diperlukan

Proses pengurutan dengan metode Shell dapat dijelaskan sebagai berikut :
Pertama-tama adalah menentukan jarak mula-mula dari data yang akan dibandingkan, yaitu N / 2. Data pertama dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 2. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j selalu lebih kecil daripada data ke-(j + N / 2).

Pada proses berikutnya, digunakan jarak (N / 2) / 2 atau N / 4. Data pertama dibandingkan dengan data dengan jarak N / 4. Apabila data pertama lebih besar dari data ke N / 4 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan

dengan jarak yang sama yaitu N / 4. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j lebih kecil daripada data ke-(j + N / 4).

Pada proses berikutnya, digunakan jarak (N / 4) / 2 atau N / 8. Demikian seterusnya sampai jarak yang digunakan adalah 1.

Algoritma metode Shell dapat dituliskan sebagai berikut :

```
1     Jarak ← N
2     Selama (Jarak > 1) kerjakan baris 3 sampai dengan 9
3     Jarak ← Jarak / 2.  Sudah ← false
4     Kerjakan baris 4 sampai dengan 8 selama Sudah = false
5     Sudah ← true
6     j ← 0
7     Selama (j < N – Jarak) kerjakan baris 8 dan 9
8     Jika (Data[j] > Data[j + Jarak] maka tukar Data[j], Data[j
      + Jarak].  Sudah ← true
9     j ← j + 1
```

Untuk lebih memperjelas langkah-langkah algoritma penyisipan langsung dapat dilihat pada tabel 1. Proses pengurutan pada tabel 1 dapat dijelaskan sebagai berikut:

- Pada saat Jarak = 5, j diulang dari 0 sampai dengan 4. Pada pengulangan pertama, Data[0] dibandingkan dengan Data[5]. Karena 12<17, maka tidak terjadi penukaran. Kemudian Data[1] dibandingkan dengan Data[6]. Karena 35>23 maka Data[1] ditukar dengan Data[6]. Demikian seterusnya sampai j=4.
- Pada saat Jarak = 5/2 = 2, j diulang dari 0 sampai dengan 7. Pada pengulangan pertama, Data[0] dibandingkan dengan Data[2]. Karena 12>9 maka Data[0] ditukar dengan Data[2]. Kemudian Data[1] dibandingkan dengan Data[3] juga terjadi penukaran karena 23>11. Demikian seterusnya sampai j=7. Perhatikan untuk Jarak = 2 proses pengulangan harus dilakukan lagi karena ternyata Data[0] > Data[2]. Proses pengulangan ini berhenti bila Sudah=true.
- Demikian seterusnya sampai Jarak=1.

Tabel 1. Proses Pengurutan dengan Metode Shell

Iterasi	Data[0]	Data[1]	Data[2]	Data[3]	Data[4]	Data[5]	Data[6]	Data[7]	Data[8]	Data[9]
Awal	12	35	9	11	3	17	23	15	31	20
Jarak=5	12	35	9	11	3	17	23	15	31	20
i=6	12	**35**	9	11	3	17	**23**	15	31	20
Jarak=2	12	23	9	11	3	17	35	15	31	20
i=2	**12**	23	**9**	11	3	17	35	15	31	20
i=3	9	**23**	12	**11**	3	17	35	15	31	20
i=4	9	11	**12**	23	**3**	17	35	15	31	20
i=5	9	11	3	**23**	12	**17**	35	15	31	20
i=7	9	11	3	17	12	**23**	35	**15**	31	20
i=8	9	11	3	17	12	15	**35**	23	**31**	20
i=9	9	11	3	17	12	15	31	**23**	35	**20**
i=2	**9**	11	**3**	17	12	15	31	20	35	23
i=3	3	**11**	9	**17**	12	15	31	20	35	23
Jarak=1	3	11	9	15	12	17	31	20	35	23
i=2	3	**11**	**9**	15	12	17	31	20	35	23
i=4	3	9	11	**15**	**12**	17	31	20	35	23
i=7	3	9	11	12	15	17	**31**	**20**	35	23
i=9	3	9	11	12	15	17	20	31	**35**	**23**
i=1	**3**	**9**	11	12	15	17	20	31	23	35
i=8	3	9	11	12	15	17	20	**31**	**23**	35
i=9	3	9	11	12	15	17	20	23	**31**	**35**
Akhir	3	9	11	12	15	17	20	23	31	35

Di bawah ini merupakan prosedur yang menggunakan metode Shell.

```
void ShellSort(int N)
{
   int Jarak, i, j;
   bool Sudah;
   Jarak = N;
   while(Lompat > 1){
      Jarak = Jarak / 2;
      Sudah = false;
      while(!Sudah){
         Sudah = true;
         for(j=0; j<N-Jarak; j++){
            i = j + Jarak;
```

```
            if(Data[j] > Data[i]){
               Tukar(&Data[j], &Data[i]);
               Sudah = false;
            }
         }
      }
   }
}
```

Program 1. Prosedur Pengurutan dengan Metode Shell

C. TUGAS PENDAHULUAN

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

D. PERCOBAAN

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

```
public class ShellDemo {

    public static void shellSort(int[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        int temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
```

```
            }
        }

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

public class MainShell {
    public static void main(String[] args) {
        int A[] = {10,6,8,3,1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

Percobaan 2 : *Shell sort* secara *descending* dengan data int.

```
public class ShellDemo {

    public static void shellSort(int[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        int temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
            }
```

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

public class MainShell {
    public static void main(String[] args) {
        int A[] = {10,6,8,3,1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

Percobaan 3 : *Shell sort* secara *ascending* dengan data double.

```
public class ShellDemo {
    public static void shellSort(double[] arr) {
        int n = arr.length;
        int C,M ;

        int jarak, i, j, kondisi;
        boolean Sudah = true;
        double temp ;

        C = 0;
        M = 0;
        jarak = n;

        while (jarak >= 1) {
            jarak = jarak / 2;
            Sudah = true;
            while (Sudah) {
                Sudah = false;
                for (j = 0; j < n - jarak; j++) {
                    i = j + jarak;
                    C++;
                    if (arr[j]> arr[i]) {
                        temp = arr[j];
                        arr[j] = arr[i];
                        arr[i] = temp;
                        Sudah = true;
                    }
                }
            }
        }

    }
     public static void tampil(double data[]){
```

```
          for(int i=0;i<data.length;i++)
              System.out.print(data[i]+" ");
          System.out.println();
    }
}

public class MainShell2 {
    public static void main(String[] args) {
        double A[] = {10.3,6.2,8.4,3.6,1.1};
        ShellDemo.tampil(A);
        ShellDemo.shellSort(A);
        ShellDemo.tampil(A);
    }
}
```

PRAKTIKUM 13
ALGORITMA PENGURUTAN (QUICK SORT)

A. TUJUAN PEMBELAJARAN

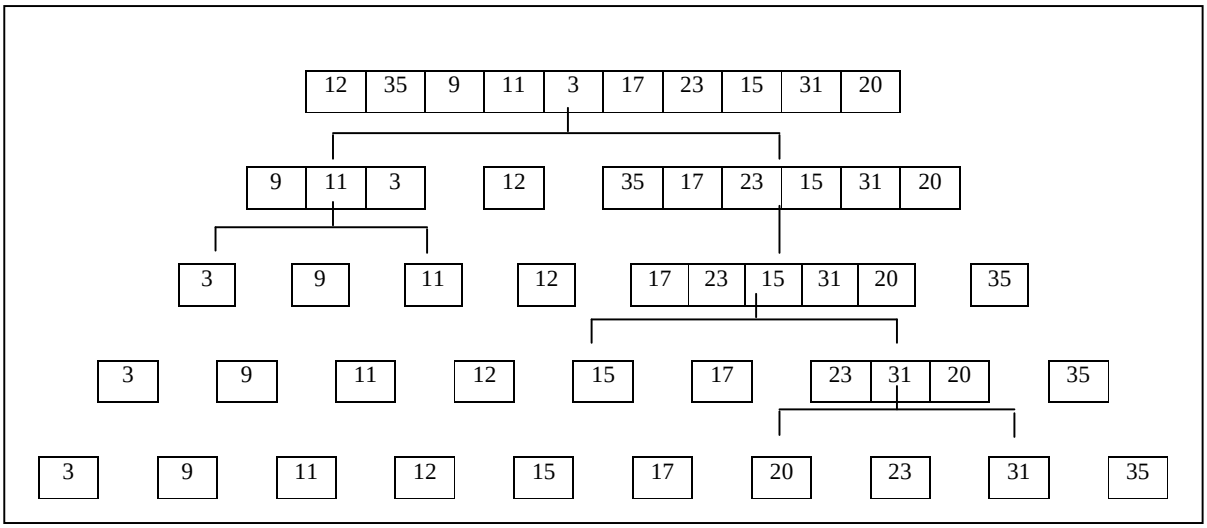
1. Memahami step by step algoritma pengurutan *quick sort.*
2. Mampu mengimplementasikan algoritma pengurutan quick sort dengan berbagai macam parameter berupa tipe data primitif atau tipe Generic.
3. Mampu mengimplementasikan algoritma pengurutan quick sort secara ascending dan descending.

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);
    }
}
```

PRAKTIKUM
ALGORITMA PENGURUTAN (MERGE SORT)

A. TUJUAN PEMBELAJARAN

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

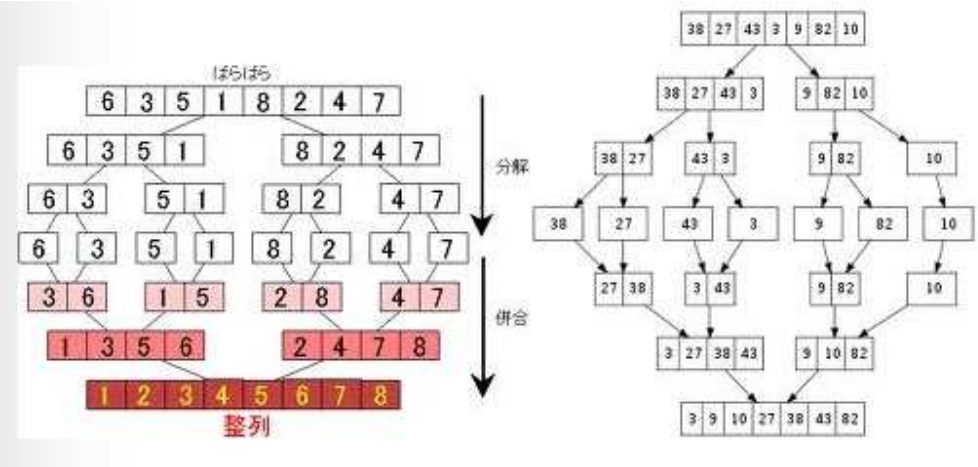
B. DASAR TEORI

Algoritma *Merge Sort*

Merge sort merupakan algoritma pengurutan dalam ilmu komputer yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data yang tidak memungkinkan untuk ditampung dalam memori komputer karena jumlahnya yang terlalu besar. Algoritma ini ditemukan oleh John von Neumann pada tahun 1945.

Algoritma pengurutan data merge sort dilakukan dengan menggunakan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok.

Setelah itu digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.



Gambar 1. Contoh pengurutan menggunakan *MergeSort*

Contoh penerapan atas sebuah larik/array dengan data yang akan diurutkan {6,3,5,1,8,2,4,7} ditunjukkan pada gambar 1. Pertama kali larik tersebut dibagi menjadi dua bagian, {6,3,5,1} dan {8,2,4,7}. Larik {6,3,5,1} dibagi menjadi dua bagian yaitu, {6,3} dan {5,1}. Larik {6,3} dibagi menjadi dua bagian yaitu, {6} dan {3}. Selanjutnya karena {6} dan {3} sudah tidak bisa dibagi lagi maka di merge dan diurutkan menjadi {3,6}. Larik {5,1} dibagi menjadi dua bagian yaitu, {5} dan {1}. Selanjutnya {5} dan {1} dilakukan merge dan diurutkan menjadi {1,5}. Larik {3,6} dan {1,5} dimerge dan diurutkan menjadi {1,3,5,6}.

Larik {8,2,4,7} dibagi menjadi dua bagian yaitu, {8,2} dan {4,7}. Larik {8,2} dibagi menjadi dua bagian yaitu, {8} dan {2}. Selanjutnya {8} dan {2} di merge dan diurutkan menjadi {2,8}. Larik {4,7} dibagi menjadi dua bagian yaitu, {4} dan {7}. Selanjutnya {4} dan {7} dilakukan merge dan diurutkan menjadi {4,7}. Larik {2,8} dan {4,7} dimerge dan diurutkan menjadi {2,4,7,8}. Larik {1,3,5,6} dan larik {2,4,7,8} dimerge dan diurutkan menjadi {1,2,3,4,5,6,7,8}.

C. TUGAS PENDAHULUAN

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

D. PERCOBAAN

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

```
public class MergeDemo {

    private int[] array;
    private int[] tempMergArr;
    private int length;

    public MergeDemo(int[] array) {
        this.array = array;
        this.length = array.length;
        this.tempMergArr = new int[length];
    }

    public void mergeSort(int lowerIndex, int higherIndex) {
        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            mergeSort(lowerIndex, middle);
            // Below step sorts the right side of the array
            mergeSort(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }

    private void mergeParts(int lowerIndex, int middle, int
higherIndex) {

        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] <= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }
    }
```

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

public class MainMerge {
    public static void main(String[] args) {
        int[] inputArr = {45, 23, 11, 89, 77, 98, 4, 28, 65, 43};
        MergeDemo mms = new MergeDemo(inputArr);
        mms.tampil();
        mms.mergeSort(0,inputArr.length-1);
        mms.tampil();
    }
}
```

Percobaan 2 : *Merge sort* secara *descending* dengan data int.

```
public class MergeDemo {

    private int[] array;
    private int[] tempMergArr;
    private int length;

    public MergeDemo(int[] array) {
        this.array = array;
        this.length = array.length;
        this.tempMergArr = new int[length];
    }

    public void mergeSort(int lowerIndex, int higherIndex) {
        if (lowerIndex < higherIndex) {
            int middle = lowerIndex + (higherIndex - lowerIndex) / 2;
            // Below step sorts the left side of the array
            mergeSort(lowerIndex, middle);
            // Below step sorts the right side of the array
            mergeSort(middle + 1, higherIndex);
            // Now merge both sides
            mergeParts(lowerIndex, middle, higherIndex);
        }
    }

    private void mergeParts(int lowerIndex, int middle, int
higherIndex) {

        for (int i = lowerIndex; i <= higherIndex; i++) {
            tempMergArr[i] = array[i];
        }
        int i = lowerIndex;
        int j = middle + 1;
        int k = lowerIndex;
```

```
        while (i <= middle && j <= higherIndex) {
            if (tempMergArr[i] >= tempMergArr[j]) {
                array[k] = tempMergArr[i];
                i++;
            } else {
                array[k] = tempMergArr[j];
                j++;
            }
            k++;
        }
        while (i <= middle) {
            array[k] = tempMergArr[i];
            k++;
            i++;
        }
    }

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

public class MainMerge {
    public static void main(String[] args) {
        int[] inputArr = {45, 23, 11, 89, 77, 98, 4, 28, 65, 43};
        MergeDemo mms = new MergeDemo(inputArr);
        mms.tampil();
        mms.mergeSort(0,inputArr.length-1);
        mms.tampil();
    }
}
```