Algoritma dan Struktur Data 2

Modul 3
Algoritma Pengurutan

Disusun oleh:

Dwi Intan Af'idah, S.T., M.Kom

PROGRAM STUDI TEKNIK INFORMATIKA
POLITEKNIK HARAPAN BERSAMA
TAHUN AJARAN 2020/2021

Daftar Isi

1 Pengurutan / *Sorting*

1.1 Target Pembelajaran

1. Mengetahui cara menerapkan algoritma pengurutan pada suatu kasus.
2. Mengetahui cara membuat kode program menggunakan algoritma pengurutan.
3. Mengetahui jenis-jenis algoritma pengurutan seperti *insertion sort, bubble sort, selection sort, shell sort, quick short, merge short.*
4. Memahami cara menentukan algoritma pencarian yang cocok digunakan pada suatu kasus.

1.2 Dasar Teori

Algoritma *sorting* merupakan salah satu tugas yang dilakukan oleh komputer. Pengurutan data yang dilakukan Algoritma *Sorting* memegang fungsi penting agar sebuah masalah dapat diselesaikan lebih cepat dan tepat.

Proses pengurutan data banyak digunakan pada proses-proses umum seperti mengurutkan data tanggal dari yang terbaru ke lama ataupun sebaliknya. Pengurutan dapat bersifat *ascending* atau *descending.*

1.2.1 Insertion sort

Salah satu algoritma sorting yang paling sederhana adalah insertion sort. Algoritma insertion sort pada dasarnya memilah data yang akan urutkan menjadi 2 bagian, yang belum diurutkan dan yang sudah diurutkan.

Elemen pertama diambil dari bagian array yang belum diurutkan dan kemudian diletakkan sesuai posisinya pada bagian lain dari array yang telah diurutkan. Langkah ini dilakukan secara berulang hingga tak ada lagi elemen tersisa pada bagian array yang belum diurutkan.

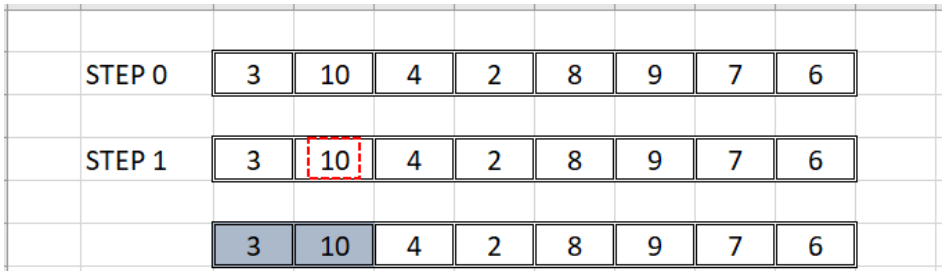
Metode insection sort merupakan metode yang mengurutkan bilangan-bilangan yang telah terbaca, dan berikutnya secara berulang akan menyisipkan bilangan-bilangan dalam array yang belum terbaca ke sisi kiri array yang telah terurut. Pengurutan dimulai dengan mengambil pada bilangan yang paling kiri.

Simulasi algoritma *insertion sort (1).*

1. Berikut data yang akan diurutkan

3	10	4	2	8	9	7	6

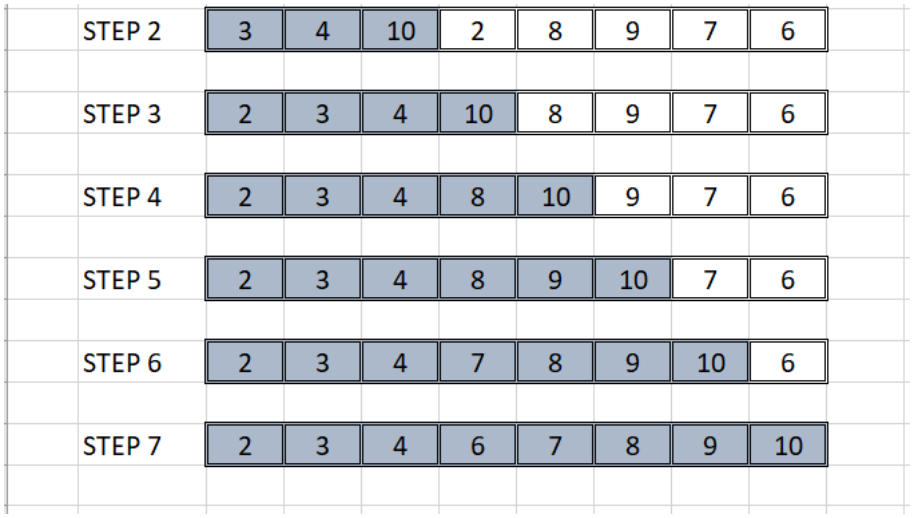
2. Cek bilangan indeks-1 (10) apakah lebih kecil dari bilangan indeks-0. Apabila lebih kecil maka ditukar. Jadi tiap bilangan indeks-1 lebih besar dari bilangan indeks-0 maka tidak ditukar.



STEP 0	3	10	4	2	8	9	7	6
STEP 1	3	10	4	2	8	9	7	6
	3	10	4	2	8	9	7	6

3. Pada kotak warna abu2 sudah dalam keadaan terurut.

4. Kemudian membandingkan lagi pada bilangan selanjutnya yaitu bilangan indeks-2 (4) dengan bilangan yang ada di sebelah kirinya. Pada kasus ini bilangan indeks-1 bergeser dan digantikan bilangan indeks-2. Lakukan langkah seperti di atas pada bilangan selanjutnya.



STEP 2	3	4	10	2	8	9	7	6
STEP 3	2	3	4	10	8	9	7	6
STEP 4	2	3	4	8	10	9	7	6
STEP 5	2	3	4	8	9	10	7	6
STEP 6	2	3	4	7	8	9	10	6
STEP 7	2	3	4	6	7	8	9	10

1.2.2 Bubble Sort

Metode gelembung (bubble sort) sering juga disebut dengan metode penukaran (exchange sort) adalah metode yang mengurutkan data dengan cara membandingkan masing-masing elemen, kemudian melakukan penukaran bila perlu. Metode ini mudah dipahami dan diprogram, tetapi bila dibandingkan dengan metode lain yang kita pelajari, metode ini merupakan metode yang paling tidak efisien.

Proses pengurutan metode gelembung ini menggunakan dua kalang. Kalang pertama melakukan pengulangan dari elemen ke 1 sampai dengan elemen ke N-1 (misalnya variable i), sedangkan kalang kedua melakukan pengulangan menurun dari elemen ke N-1 sampai elemen ke i (misalnya variable j). Pada setiap pengulangan, elemen ke j-1 dibandingkan dengan elemen ke j. Apabila data ke j-1 lebih besar daripada data ke j, dilakukan penukaran.

Tabel 1. Proses Pengurutan dengan Metode Bubble Sort

		10	6	8	3	1
i=1	j=4	10	6	8	3	1
	j=3	10	6	8	1	3
	j=2	10	6	1	8	3
	j=1	10	1	6	8	3
i=2	j=4	1	10	6	8	3
	j=3	1	10	6	3	8
	j=2	1	10	3	6	8
i=3	j=4	1	3	10	6	8
	j=3	1	3	10	6	8
i=4	j=4	1	3	6	10	8
	Akhir	1	3	6	8	10

Untuk lebih memperjelas langkah-langkah algoritma gelembung dapat dilihat pada Tabel 1. Proses pengurutan pada Tabel 1 dapat dijelaskan sebagai berikut:

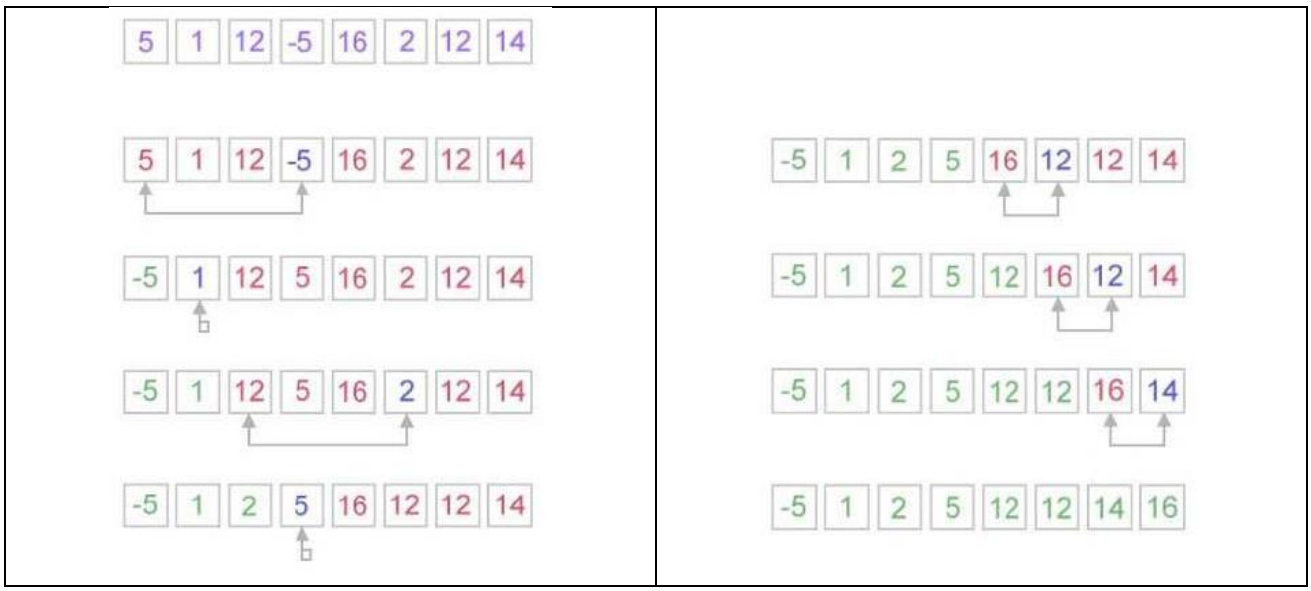
- Pada saat i=1, nilai j diulang dari 4 sampai dengan 1. Pada pengulangan pertama Data[4] dibandingkan Data[3], karena 1 < 3 maka Data[4] dan Data[3] ditukar. Pada pengulangan kedua Data[3] dibandingkan Data[2], karena 1 > 8 maka Data[3] dan Data[2] ditukar. Demikian seterusnya sampai j=1.
- Pada saat i=2, nilai j diulang dari 4 sampai dengan 2. Pada pengulangan pertama Data[4] dibandingkan Data[3], karena 3 < 8, Data[4] dan Data[3] ditukar. Demikian seterusnya sampai j=2.
- Dan seterusnya sampai dengan i=4.

1.2.3 Selection Sort

Ide utama adalah pada data indeks ke-0, dibandingkan dengan data sesudahnya untuk mencari elemen yang paling kecil, selanjutnya elemen terkecil tersebut ditukar dengan elemen pada indeks ke-0. Selanjutnya data indeks ke-1, dibandingkan dengan data sesudahnya untuk mencari elemen yang paling kecil, selanjutnya elemen terkecil tersebut ditukar dengan elemen pada indeks ke-1, dan seterusnya sampai data terurut secara ascending.

Contoh terdapat data 5, 1, 12, -5, 16, 2, 12, 14. Data acuan pada indek ke-0 yaitu 5 dibandingkan dengan data sesudahnya untuk mencari elemen terkecil. Elemen terkecil setelah 5 adalah -5, sehingga 5 ditukar dengan -5, sehingga data menjadi -5, 1, 12, 5, 16, 2, 12, 14. Data acuan berikutnya adalah indek ke-1 yaitu 1 dibandingkan dengan data sesudahnya untuk mencari elemen terkecil. Elemen terkecil setelah 1 ternyata adalah 1, sehingga 1 ditukar dengan 1, sehingga data menjadi -5, 1, 12, 5, 16, 2, 12, 14 dan seterusnya.

Simulasi algortima *selection sort*



1.2.4 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:

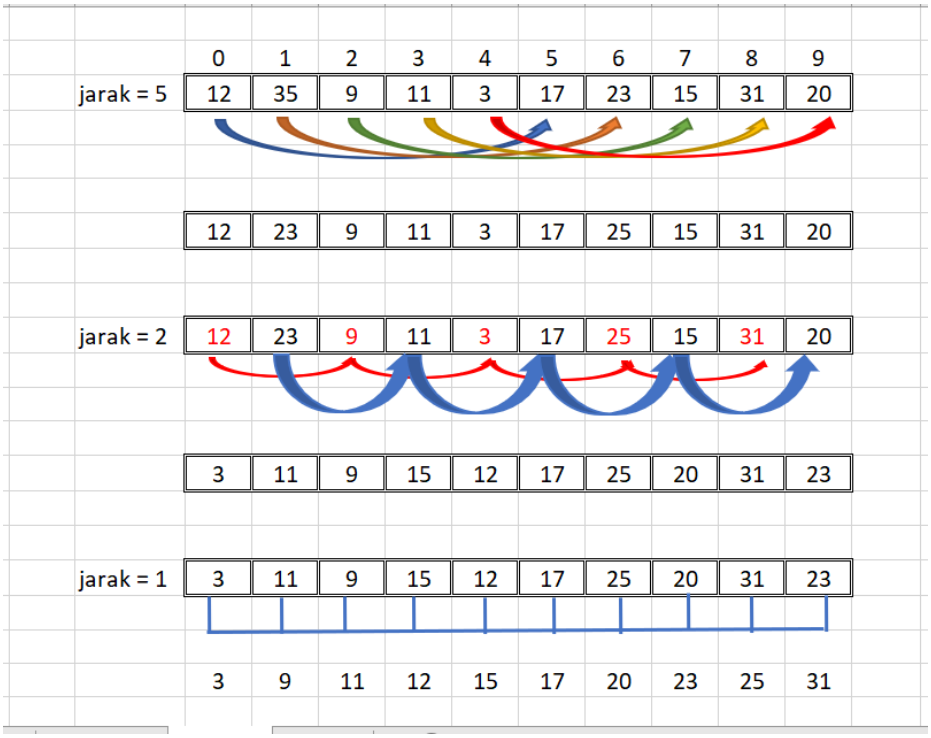
Pertama-tama adalah menentukan jarak mula-mula dari data yang akan dibandingkan, yaitu N / 2. Data pertama dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama 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).

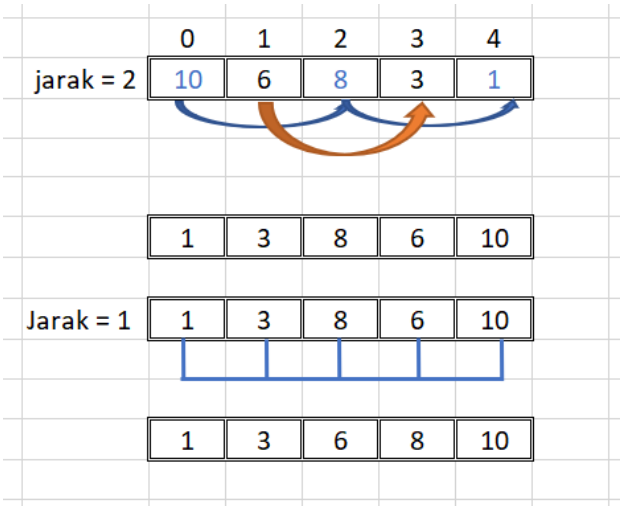
Pada proses berikutnya, digunakan jarak (N / 4) / 2 atau N / 8. Demikian seterusnya sampai jarak yang digunakan adalah 1.

Untuk lebih memperjelas langkah algoritma penyisipan langsung dapat dilihat pada Gambar 1.

- 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 Jarak=1.



Gambar 1. Algoritma Shell Sort



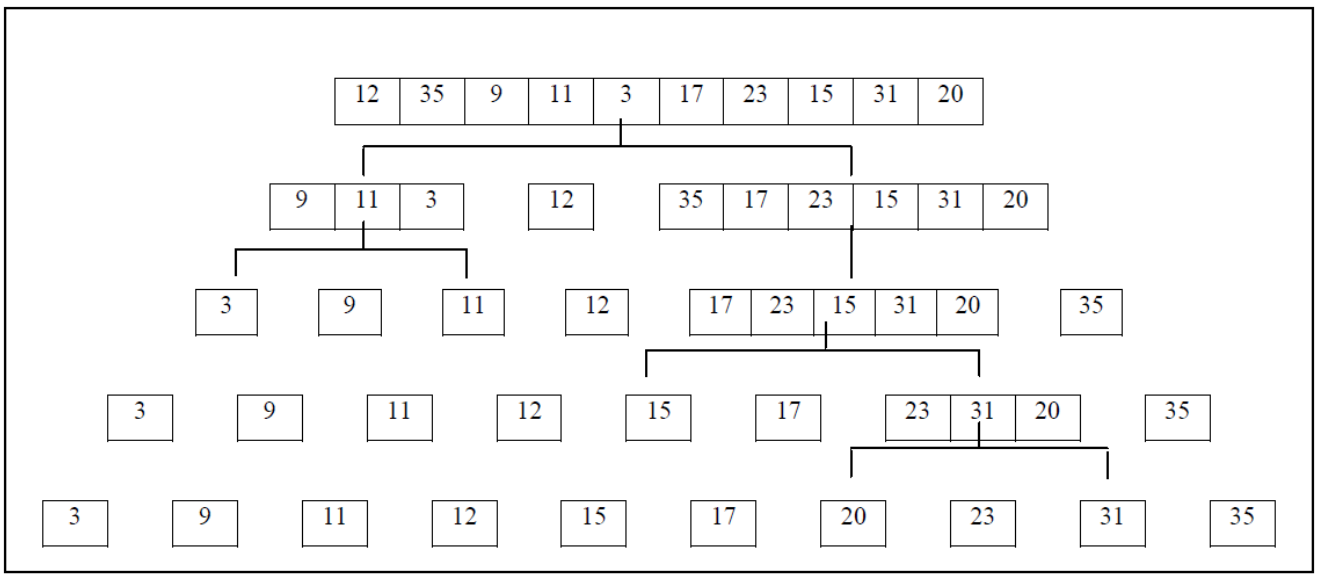
Gambar 2. Algoritma Shell Sort dengan length=5

1.2.5 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 3.



Gambar 3. Ilustrasi Algoritma Quick Short

Gambar 3 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 3, 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 kita 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 penempatan 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

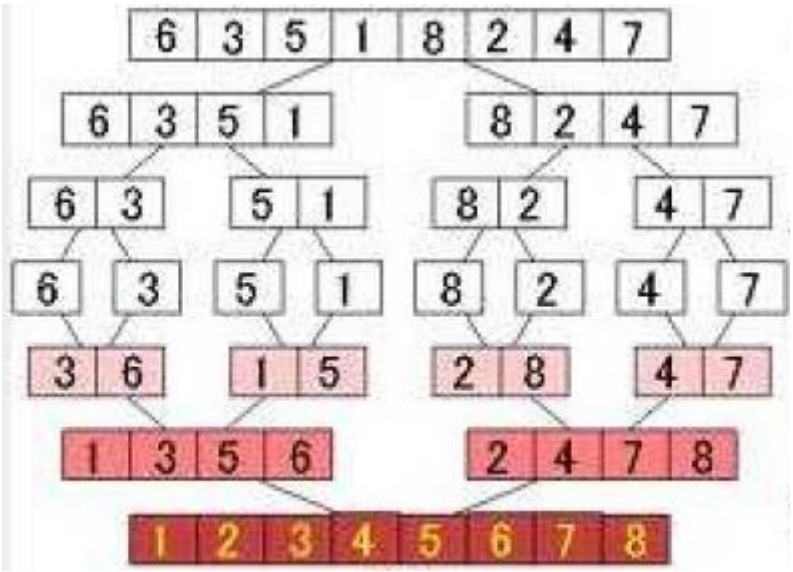
3	11	9		12		35	17	23	15	31	20

1.2.6 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 4. Ilustrasi Algoritma Merge Sort

Contoh penerapan atas sebuah larik/array dengan data yang akan diurutkan {6,3,5,1,8,2,4,7} ditunjukkan pada gambar 4. 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}.

1.3 Latihan

1.3.1 Insertion Sort

Data:

3	10	4	2	8	9	7	6

Kode program

```
1
2   package Sorting;
3
4   public class A_insertionsort {
5
6       public static void insertionSort(int[] A) {
7           for (int i = 1; i < A.length; i++) { //i=1 //i=2 //i=3
8               int key = A[i]; //A[1]=10 //A[2]=4  //A[3]=2
9               int j = i - 1; //j=0 //j=1 //j=2
10              while ((j >= 0) && (A[j] > key)) { //0>=0&A[0]3>7-F
11                                                 //1>=0&A[1]10>4-T /0>=0&A[0]3>4-F
12                                                 //2>=0&A[2]10>2-T  /1>=0&A[1]4>2-T  /0>=0&A[0]3>2-T /-1>=0-F
13                  A[j + 1] = A[j]; //A[2]=A[1]=10     //A[3]=A[2]=10 /A[2]=A[1]=4 /A[1]=A[0]=3
14                  j--; //j=0      //j=1 /j=0 /j=-1
15              }
16              A[j + 1] = key; //A[1]=10  //A[1]=4     //A[0]=2
17          } //3,10,  4,2,8,9,7,6   //3,4,10,  2,8,9,7,6   //2,3,4,10, 8,9,7,6
18      }
19
20      public static void tampil(int data[]) {
21          for (int i = 0; i < data.length; i++) {
22              System.out.print(data[i] + " ");
23          }
24      System.out.println();
25      }
26
27      public static void main(String[] args) {
28          int A[] = {3, 10, 4, 2, 8, 9, 7, 6};
29          A_insertionsort.tampil(A);
30          A_insertionsort.insertionSort(A);
31          A_insertionsort.tampil(A);
32      }
33  }
```

Output:

```
run-single:
3 10 4 2 8 9 7 6
2 3 4 6 7 8 9 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3.2 Bubble Sort

Kode Program

```
1
2   package Sorting;
3
4   public class B_bubblesort {
5       public static void bubbleSort(int[] A) {
6           int i=1, j, n = A.length;
7           int temp;
8           while (i<n){     //1<5
9               j = n-1 ;    //j=5-1=4
10              while(j>=i){     //4>=1-T
11                  if (A[j-1]>A[j]){ //A[3]>A[4]-3>1-T
12                      temp = A[j]; //temp=1
13                      A[j] = A[j-1]; //A[4]=A[3]=3
14                      A[j-1] = temp; //A[3]=1
15                  }
16                  j = j - 1; //j=3
17              }
18              i = i + 1; //i=i+1
19          }
20      }
21
22      public static void tampil(int data[]) {
23          for (int i = 0; i < data.length; i++) {
24              System.out.print(data[i] + " ");
25          }
26          System.out.println();
27      }
28
29
30      public static void main(String[] args) {
31          int A[] = {10,6,8,3,1};
32          B_bubblesort.tampil(A);
33          B_bubblesort.bubbleSort(A);
34          B_bubblesort.tampil(A);
35      }
36  }
```

Output

```
run-single:
10 6 8 3 1
1 3 6 8 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3.3 Selection Sort

Kode Program

```
1   package Sorting;
2
3   public class C_selectionsort {
4       public static void selectionSort(int[] A) {
5           int smallIndex;
6           int pass, j, n = A.length;
7           int temp;
8
9           // index of smallest element in the sublist
10          for (pass = 0; pass < n - 1; pass++) { //pass=0; 0<4-T
11              smallIndex = pass;                    //0
12              for (j = pass + 1; j < n; j++){       //j=1; 1<5  /j=2; 2<5
13                  if (A[j]< A[smallIndex]) {        //A[1]<A[0]-6<10-T  /A[2]<A[1]-8<6-F
14                      smallIndex = j; //1  /2
15                  }
16              }
17              // tukar nilai terkecil dengan array[pass]
18              temp = A[pass]; //
19              A[pass] = A[smallIndex];
20              A[smallIndex] = temp;
21          }
22      }
23
24      public static void tampil(int data[]) {
25          for (int i = 0; i < data.length; i++) {
26              System.out.print(data[i] + " ");
27          }
28          System.out.println();
29      }
30
31      public static void main(String[] args) {
32          int A[] = {10,6,8,3,1};
33          C_selectionsort.tampil(A);
34          C_selectionsort.selectionSort(A);
35          C_selectionsort.tampil(A);
36      }
37  }
38
```

Output

```
run-single:
10 6 8 3 1
1 3 6 8 10
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3.4 Shell Sort

Kode Program

```
1   package Sorting;
2
3   public class D_shellsort {
4       public static void shellSort(int[] arr) {
5           int n = arr.length;
6           int C,M ;
7           int jarak, i, j, kondisi;
9           boolean Sudah = true;
10          int temp ;
11          C = 0;
12          M = 0;
13          jarak = n;
14
15          while (jarak >= 1) {
16              jarak = jarak / 2;
17              Sudah = true;
18              while (Sudah) {
19                  Sudah = false;
20                  for (j = 0; j < n - jarak; j++) {
21                      i = j + jarak;  //i=0+2 = 2
22                      C++;
23                      if (arr[j]> arr[i]) {
24                          temp = arr[j];
25                          arr[j] = arr[i];
26                          arr[i] = temp;
27                          Sudah = true;
28                      }
29                  }
30              }
31          }
32
33          public static void tampil(int data[]){
34              for(int i=0;i<data.length;i++)
35                  System.out.print(data[i]+" ");
36              System.out.println();
37          }
38
39
40          public static void main(String[] args) {
41              int A[] = {12,35,9,11,3,17,23,15,31,20};
42              D_shellsort.tampil(A);
43              D_shellsort.shellSort(A);
44              D_shellsort.tampil(A);
45          }
46      }
```

Output:

```
run-single:
12 35 9 11 3 17 23 15 31 20
3 9 11 12 15 17 20 23 31 35
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3.5 Quick Sort

Kode Program

```
1   package Sorting;
2
3   public class E_quicksort {
4       private static int partition(int[] A, int p, int r) {
5           int i, j;
6           double pivot;
7
8           pivot = A[p];
9           i = p - 1;
10          j = r + 1;
11          for (;;) {
12              do {
13                  i++;
14              } while (A[i] < pivot);
15              do {
16              j--;
17              } while (A[j] > pivot);
18
19              if (i < j) {
20                  int temp = A[i];
21                  A[i] = A[j];
22                  A[j] = temp;
23              } else {
24                  return j;
25              }
26          }
27      }
28
29      public static void quickSort(double[] A, int p, int r) {
30      int q;
31      if (p < r) {
32          q = partition(A,p,r);
33          quickSort(A,p, q);
34          quickSort(A,q + 1, r);
35      }
36      }
37
38      public static void tampil(double data[]){
39          for(int i=0;i<data.length;i++)
40              System.out.print(data[i]+" ");
41          System.out.println();
42      }
43
44      public static void main(String[] args) {
45          double A[] = {12,35,9,11,3,17,23,15,31,20};
46          E_quicksort.tampil(A);
47          E_quicksort.quickSort(A,0,A.length-1);
48          E_quicksort.tampil(A);
49      }
50  }
```

Output

```
run-single:
12 35 9 11 3 17 23 15 31 20
3 9 11 12 15 17 20 23 31 35
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.3.6 Merge Sort

Kode Program

```
package Sorting;

public class F_mergeshort {
    private int[] array;
    private int[] tempMergArr;
    private int length;

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

Output

```
run-single:
45 23 11 89 77 98 4 28 65 43
4 11 23 28 43 45 65 77 89 98
BUILD SUCCESSFUL (total time: 0 seconds)
```

1.4 Tugas Praktikum 4

- **Buatlah kode program sorting untuk data di bawah ini**

25	7	9	13	3

- **Cetaklah nama lengkap pada setiap program yang dibuat**
- **Buatlah gambaran ilustrasi dari masing-masing algortima.**

1. Program menggunakan algoritma *insertion sort.*
2. Program menggunakan algoritma *bubble sort.*
3. Program menggunakan algoritma *selection sort.*
4. Program menggunakan algoritma *shell sort.*
5. Program menggunakan algoritma *quick short.*
6. Program menggunakan algoritma *merge short.*
7. Program menggunakan satu algoritma sorting dan satu algoritma searching.