Materi Sesi 5

Pengurutan (Sorting)

Tujuan

1. Menunjukkan beberapa algoritma dalam Pengurutan
2. Menunjukkan bahwa pengurutan merupakan suatu persoalan yang bisa diselesaikan dengan sejumlah algoritma yang berbeda satu sama lain lengkap dengan kelebihan dan kekurangannya
3. Dapat memilih algoritma yang paling sesuai untuk menyelesaikan suatu permasalahan pemrograman

---

Pengurutan data (*sorting*) didefinisikan sebagai suatu proses untuk menyusun kembali humpunan obyek menggunakan aturan tertentu. Mnurut Microsoft Book-shelf, definisi algoritma pengurutan adalah algoritma untuk meletakkan kumpulan elemen data ke dalam urutan tertentu berdasarkan satu atau beberapa kunci dalam tiap-tiap elemen.

Ada dua macam urutan yang biasa digunakan dalam proses pengurutan yaitu

- urut naik (*ascending*) yaitu dari data yang mempunyai nilai paling kecil sampai paling besar
- urut turun (*descending*) yaitu data yang mempunyai nilai paling besar sampai paling kecil.

Contoh : data bilangan 5, 2, 6 dan 4 dapat diurutkan naik menjadi 2, 4, 5, 6 atau diurutkan turun menjadi 6, 5, 4, 2. Pada data yang bertipe char, nilai data dikatakan lebih kecil atau lebih besar dari yang lain didasarkan pada urutan relatif (*collating sequence*) seperti dinyatakan dalam tabel ASCII (Lampiran)

Keuntungan dari data yang sudah dalam keadaan terurutkan antara lain :

- data mudah dicari (misalnya dalam buku telepon atau kamus bahasa), mudah untuk dibetulkan, dihapus, disisipi atau digabungkan. Dalam keadaan terurutkan, kita mudah melakukan pengeekan apakah ada data yang hilang
- melakukan komppilasi program komputer jika tabel-tabel simbol harus dibentuk
- mempercepat proses pencarian data yang harus dilakukan berulang kali.

Data yang diurutkan sangat bervariasi, dalam hal jumlah data maupun jenis data yang akan diurutkan. Tidak ada algoritma terbaik untuk setiap situasi yang kita hadapi, bahkan cukup sulit untuk menentukan algoritma mana yang paling baik untuk situasi tertentu karena ada beberapa faktor yang mempengaruhi efektifitas algoritma pengurutan. Beberapa faktor yang berpengaruh pada efektifitas suatu algoritma pengurutan antara lain:

- banyak data yang diurutkan
- kapasitas pengingat apakah mampu menyimpan semua data yang kita miliki
- tempat penyimpanan data, misalnya piringan, pita atau kartu, atau media penyimpan yang lain.

Pemilihan algoritma sangat ditentukan oleh struktur data yang digunakan. Metode pengurutan yang digunakan dapat diklasifikasikan menjadi dua katagori yaitu :

- pengurutan internal, yaitu pengurutan dengan menggunakan larik (*array*). Larik tersimpan dalam memori utama komputer
- pengurutan eksternal, yaitu pengurutan dengan menggunakan berkas (*sequential access file*). Berkas tersimpan dalam pengingat luar, misalnya cakram atau pita magnetis.

Untuk menggambarkan pengurutan dengan larik, bisa kita bayangkan semua kartu terletak di hadapan kita sehingga semua kartu terlihat dengan jelas nomornya. Pada penyusunan kartu sebagai sebuah berkas, kita bayangkan semua kartu kita tumpuk sehingga hanya kartu bagian atas saja yang bisa kita lihat nomornya.

Pada bab ini akan dijelaskan beberapa metode pengurutan beserta analisanya.

1. Deklarasi Larik

Pada pengurutan larik, ada beberapa aspek yang perlu dipertimbangkan, antara lain aspek menyangkut kapasitas pengingat yang ada dan aspek waktu, yaitu waktu yang diperlukan untuk melakukan permutasi sehingga semua elemen akhirnya menjadi terurutkan.

Deklarasi larik yang digunakan adalah larik dimensi satu (*vektor*) dengan elemennya bertipe integer.

```
#define  Max 100;
int Data[Max];
```

Pada deklarasi diatas, `Max` adalah banyaknya elemen vektor. Anda bisa mengubah nilai konstantaa `Max` sesuai kebutuhan. Indeks larik dimulai dari 0. Data yang sebenarnya disimpan mulai dari indeks 0.

Selain deklarasi diatas, proses yang juga selalu digunakan pada algoritma pengurutan adalah proses menukarkan elemen. Di bawah ini satu prosedur sederhana untuk menukarkan nilai dua buah elemen A dan B.

```
void Tukar (int *a, int *b)
{
      int temp;
      temp = *a;
      *a = *b;
      *b = temp;
}
```

Program 1 Prosedur Penukaran 2 Bilangan

Prosedur tukar diatas akan digunakan pada beberapa prosedur pengurutan yang akan dijelaskan dibawah ini

2. Metode Penyisipan Langsung (*Straight Insertion Sort*)

Proses pengurutan dengan metode penyisipan langsung dapat dijelaskan sebagai berikut :

Data dicek satu per satu mulai dari yang kedua sampai dengan yang terakhir. Apabila ditemukan data yang lebih kecil daripada data sebelumnya, maka data tersebut disisipkan pada posisi yang sesuai. Akan lebih mudah apabila membayangkan pengurutan kartu. Pertama-tama anda meletakkan kartu-kartu tersebut di atas meja, kemudian melihatnya dari kiri ke kanan. Apabila kartu di sebelah kanan lebih kecil daripada kartu di sebelah kiri, maka ambil kartu tersebut dan sisipkan di tempat yang sesuai.

Algoritma penyisipan langsung dapat dituliskan sebagai berikut :

1 i ← 1
2 selama (i < N) kerjakan baris 3 sampai dengan 9
3 x ← Data[i]
4 j ← i – 1
5 selama (x < Data[j]) kerjakan baris 6 dan 7
6 Data[j + 1] ← Data[j]
7 j ← j – 1

8 Data[j+1] ← x

9 i ← i + 1

Untuk lebih memperjelas langkah-langkah algoritma penyisipan langsung dapat dilihat pada tabel 6.1. Proses pengurutan pada tabel 6.1 dapat dijelaskan sebagai berikut:

- Pada saat i=1, x sama dengan Data[1] = 35 dan j=0. Karena Data[0] = 12 dan 35 > 12 maka proses dilanjutkan untuk i=2.
- Pada saat i=2, x = Data[2] = 9 dan j=1. Karena Data[1] = 35 dan 9 < 35, maka dilakukan pergeseran sampai ditemukan data yang lebih kecil dari 9. Hasil pergeseran ini, Data[1] = 12 dan Data[2] = 35 sedangkan Data[0] = x = 9.
- Pada saat i=3, x = Data[3] = 11 dan j=3. Karena Dataa[2] = 35 dan 11 < 35, maka dilakukan pergeseran sampai ditemukan data yang lebih kecil dari 11. Hasil pergeseran ini, Data[2] = 12 dan Data[3] = 35 sedangkan Data[1] = x = 11.
- Dan seterusnya.

Tabel 1 Proses Pengurutan dengan Metode Penyisipan Langsung

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
i=1	12	**35**	9	11	3	17	23	15	31	20
i=2	12	35	**9**	11	3	17	23	15	31	20
i=3	9	12	35	**11**	3	17	23	15	31	20
i=4	9	11	12	35	**3**	17	23	15	31	20
i=5	3	9	11	12	35	**17**	23	15	31	20
i=6	3	9	11	12	17	35	**23**	15	31	20
i=7	3	9	11	12	17	23	35	**15**	31	20
i=8	3	9	11	12	15	17	23	35	**31**	20
i=9	3	9	11	12	15	17	23	31	35	**20**
Akhir	3	9	11	12	15	17	20	23	31	35

Di bawah ini merupakan prosedur yang menggunakan metode penyisipan langsung.

```
void StraighInsertSort()
{
     int i, j, x;
     for(i=1; i<Max; i++){
        x = Data[i];
        j = i - 1;
        while (x < Data[j]){
```

```
            Data[j+1] = Data[j];
            j--;
         }
         Data[j+1] = x;
      }
}
```

Program 2 Prosedur Pengurutan dengan Metode Penyisipan Langsung

Dari algoritma dan prosedur diatas, jumlah perbandingan (=*C*) minimum, rata-rata dan maksimum pada metode penyisipan langsung adalah

$C_{min} = N - 1$

$C_{rata\text{-}rata} = (N^2 + N + 2) / 4$

$C_{max} = (N^2 + N - 2) / 2$

Jumlah perbandingan minimum terjadi jika data sudah dalam keadaan urut, sebaliknya jumlah perbandingan maksimum terjadi jika data dalam keadaan urut terbalik.

Cara menghitung C_{min} adalah dengan melihat kalang paling luar yaitu i, pengulangan ini dimulai dari 1 sampai dengan N-1 atau sama dengan N-1

Cara menghitung C_{max} dengan menganggap selalu terjadi pergeseran. Kalang dalam (while) diatas akan melakukan pengulangan dari 0 sampai dengan i. Apabila hal ini dilakukan sebanyak N-1 kali, maka C_{max} adalah jumlah dari deret aritmetik 2, 3, 4, ..., N atau (N – 1) / 2 . (2 + N)

Cara menghitung $C_{rata\text{-}rata}$ adalah dengan menjumlahkan C_{min} dan C_{max} dan dibagi dengan 2.

Jumlah penggeseran (=*M*) minimum, rata-rata dan maksimum untuk metode penyisipan langsung adalah :

$M_{min} = 2(N - 1)$

$M_{rata\text{-}rata} = (N^2 + 7N - 8) / 4$

$M_{max} = (N^2 + 3N - 4) / 2$

a. Metode Penyisipan Biner (*Binary Insertion Sort*)

Metode ini merupakan pengembangan dari metode penyisipan langsung. Dengan cara penyisipan langsung, perbandingan selalu dimulai dari elemen pertama (data ke-0), sehingga untuk menyisipkan elemen ke i kita harus melakukan perbandingan sebanyak i-1 kali. Ide dari metode ini didasarkan pada kenyataan bahwa pada saat menggeser data ke-i, data ke 0 s/d i-1 sebenarnya sudah dalam keadaan terurut. Sebagai contoh pada tabel 6.1 diatas, pada saat i=4, data ke 0 s/d 3 sudah dalam keadaan urut : 3, 9, 12, 35.

Dengan demikian posisi dari data ke-i sebenarnya dapat ditentukan dengan pencarian biner.

Misalnya pada saat i = 7, data yang akan dipindah adalah 15 sedangkan data di sebelah kiri 15 adalah sebagai berikut :

3	9	11	12	17	23	35	**15**

Pertama-tama dicari data pada posisi paling tengah diantara data diatas. Data yang terletak di tengah adalah data ke-3, yaitu 12. Karena 12 < 15, berarti 15 harus disisipkan di sebelah kanan 12. Oleh karena itu, proses pencarian dlanjutkan lagi untuk data berikut

17	23	35

Dari hasil ini, didapat data tengahnya adalah data 23. Karena 15 < 23, berarti 15 harus disisipkan di sebelah kiri 23. Proses dilanjutkan kembali untuk data

17

Karena 17 > 15, berarti 15 harus disisipkan di sebelah kiri 17

Algoritma penyisipan biner dapat dituliskan sebagai berikut :

```
1   i ← 1
2   selama (i < N) kerjakan baris 3 sampai dengan 14
3   x ← Data[i]
4   l ← 0
5   r ← i – 1
6   selama (l<=r) kerjakan baris 7 dan 8
7   m ← (l + r) / 2
8   jika (x < Data[m]) maka r ← m – 1, jika tidak l ← m + 1
9   j ← i – 1
10  selama ( j >=l) kerjakan baris 11 dan 12
11  Data[j+1] ← Data[j]
12  j ← j – 1
13  Data[l] ← x
14  l ← i + 1
```

Di bawah ini merupakan prosedur yang menggunakan metode penyisipan biner.

```
void BinaryInsertSort()
{
     int i, j, l, r, m, x;
     for (i=1; i<Max; i++){
        x = Data[i];
        l = 0;
        r = i - 1;
        while(l <= r){
           m = (l + r) / 2;
           if(x < Data[m])
              r = m - 1;
           else
              l = m + 1;
        }
        for(j=i-1; j>=l; j--)
           Data[j+1] = Data[j];
        Data[l]=x;
     }
}
```

Program 3 Prosedur Pengurutan dengan Metode Penyisipan Biner

Dari algoritma dan prosedur diatas, jumlah perbandingan (=*C*) untuk metode penyisipan biner adalah :

$C = \sum \lceil {}^{2}\log(i) \rceil$

Sedangkan jumlah penggeseran (=*M*) untuk metode penyisipan biner sama dengan metode penyisipan langsung.

b. Metode Seleksi (*Selection Sort*)

Metode seleksi melakukan pengurutan dengan cara mencari data yang terkecil kemudian menukarkannya dengan data yang digunakan sebagai acuan atau sering dinamakan pivot.

Proses pengurutan dengan metode seleksi dapat dijelaskan sebagai berikut : langkah pertama dicari data terkecil dari data pertama sampai data terakhir. Kemudian data terkecil ditukar dengan data pertama. Dengan demikian, data pertama sekarang mempunyai nilai paling kecil dibanding data yang lain. Langkah kedua, data terkecil kita

cari mulai dari data kedua sampai terakhir. Data terkecil yang kita peroleh ditukar dengan data kedua dan demikian seterusnya sampai semua elemen dalam keadaan terurutkan.

Algoritma seleksi dapat dituliskan sebagai berikut :

1 i ← 0
2 selama (i < N-1) kerjakan baris 3 sampai dengan 9
3 k ← i
4 j ← i + 1
5 Selama (j < N) kerjakan baris 6 dan 7
6 Jika (Data[k] > Data[j]) maka k ← j
7 j ← j + 1
8 Tukar Data[i] dengan Data[k]
9 i ← i + 1

Untuk lebih memperjelas langkah-langkah algoritma seleksi dapat dilihat pada tabel 6.2. Proses pengurutan pada tabel 6.2 dapat dijelaskan sebagai berikut:

- Pada saat i=0, data terkecil antara data ke-1 s/d ke-9 adalah data ke-4, yaitu 3, maka data ke-0 yaitu 12 ditukar dengan data ke-4 yaitu 3.
- Pada saat i=1, data terkecil antara data ke-2 s/d ke-9 adalah data ke-2, yaitu 9, maka data ke-1 yaitu 35 ditukar dengan data ke-2 yaitu 9.
- Pada saat i=2, data terkecil antara data ke-3 s/d ke-9 adalah data ke-3, yaitu 11, maka data ke-2 yaitu 35 ditukar dengan data ke-3 yaitu 11.
- Pada saat i=3, data terkecil antara data ke-4 s/d ke-9 adalah data ke-4, yaitu 12, maka data ke-3 yaitu 35 ditukar dengan data ke-4 yaitu 12.
- Dan seterusnya.

Tabel 6.2 Proses Pengurutan dengan Metode Seleksi

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
i=0	**12**	35	9	11	**3**	17	23	15	31	20
i=1	3	**35**	**9**	11	12	17	23	15	31	20
i=2	3	9	**35**	**11**	12	17	23	15	31	20
i=3	3	9	11	**35**	**12**	17	23	15	31	20
i=4	3	9	11	12	**35**	17	23	**15**	31	20
i=5	3	9	11	12	15	**17**	23	35	31	20
i=6	3	9	11	12	15	17	**23**	35	31	**20**
i=7	3	9	11	12	15	17	20	**35**	31	**23**
i=8	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 seleksi.

```
void SelectionSort()
{
      int i, j, k;
      for(i=0; i<Max-1;i++){
            k = i;
            for (j=i+1; j<Max; j++)
                  if(Data[k] > Data[j])
                        k = j;
            Tukar(&Data[i], &Data[k]);
      }
}
```

Program 4 Prosedur Pengurutan dengan Metode Seleksi

Dari algoritma dan prosedur diatas, jumlah perbandingan (=*C*) metode seleksi adalah

$C = N(N - 1) / 2$

Jumlah penukaran (=*M*) pada metode seleksi tergantung keadaan datanya. Penukaran minimum terjadi bila data sudah dalam keadaan urut, sebaliknya jumlah penukaran maksimum terjadi bila data dalam keadaan urut terbalik. Jumlah penukaran minimum dan maksimum dapat dirumuskan sebagai berikut :

$M_{min} = 3(N - 1)$

$M_{max} = \lfloor N^2 / 4 \rfloor + 3(N - 1)$

c. Metode Gelembung (*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 2 sampai dengan elemen ke N-1 (misalnya variable i), sedangkan kalang kedua melakukan pengulangan menurun dari elemen ke N 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.

Algoritma gelembung dapat dituliskan sebagai berikut :

1 i ← 0

2 selama (i < N-1) kerjakan baris 3 sampai dengan 7

3 j ← N - 1

4 Selama (j >= i) kerjakan baris 5 sampai dengan 7

5 Jika (Data[j-1] > Data[j]) maka tukar Data[j-1] dengan Data[j]

6 j ← j – 1

7 i ← i + 1

Untuk lebih memperjelas langkah-langkah algoritma gelembung dapat dilihat pada tabel 6.3. Proses pengurutan pada tabel 6.3 dapat dijelaskan sebagai berikut:

- Pada saat i=1, nilai j diulang dari 9 sampai dengan 1. Pada pengulangan pertama Data[9] dibandingkan Data[8], karena 20 < 31 maka Data[9] dan Data[8] ditukar. Pada pengulangan kedua Data[8] dibandingkan Data[7], karena 20 > 15 maka proses dilanjutkan. Demikian seterusnya sampai j=1.
- Pada saat i=2, nilai j diulang dari 9 sampai dengan 2. Pada pengulangan pertama Data[9] dibandingkan Data[8], karena 31 > 20 maka proses dilanjutkan. Pada pengulangan kedua Data[8] dibandingkan Data[7], karena 20 < 23 Data[8] dan Data[7] ditukar. Demikian seterusnya sampai j=2.
- Pada saat i=3, nilai j diulang dari 9 sampai dengan 3. Pada pengulangan pertama Data[9] dibandingkan Data[8], karena 31 > 23 maka proses dilanjutkan. Pada pengulangan kedua Data[8] dibandingkan Data[7], karena 23 > 20 maka proses dilanjutkan. Demikian seterusnya sampai j=3.
- Dan seterusnya sampai dengan i=8.

Tabel 3 Proses Pengurutan dengan Metode Gelembung

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
I=1; j=9	12	35	9	11	3	17	23	15	**31**	**20**
j=8	12	35	9	11	3	17	23	**15**	**20**	31
j=7	12	35	9	11	3	17	**23**	**15**	20	31
j=6	12	35	9	11	3	**17**	**15**	23	20	31
j=5	12	35	9	11	**3**	**15**	17	23	20	31
j=4	12	35	9	**11**	**3**	15	17	23	20	31
j=3	12	35	**9**	**3**	11	15	17	23	20	31
j=2	12	**35**	**3**	9	11	15	17	23	20	31
j=1	**12**	**3**	35	9	11	15	17	23	20	31
i=2; j=9	3	12	35	9	11	15	17	23	**20**	**31**
j=8	3	12	35	9	11	15	17	**23**	**20**	31
j=7	3	12	35	9	11	15	**17**	**20**	23	31

j=6	3	12	35	9	11	**15**	**17**	20	23	31
j=5	3	12	35	9	**11**	**15**	17	20	23	31
j=4	3	12	35	**9**	**11**	15	17	20	23	31
j=3	3	12	**35**	**9**	11	15	17	20	23	31
j=2	3	**12**	**9**	35	11	15	17	20	23	31
i=3; j=9	3	9	12	35	11	15	17	20	**23**	**31**
j=8	3	9	12	35	11	15	17	**20**	**23**	31
j=7	3	9	12	35	11	15	**17**	**20**	23	31
j=6	3	9	12	35	11	**15**	**17**	20	23	31
j=5	3	9	12	35	**11**	**15**	17	20	23	31
j=4	3	9	12	**35**	**11**	15	17	20	23	31
j=3	3	9	**12**	**11**	35	15	17	20	23	31
i=4; j=9	3	9	11	12	35	15	17	20	**23**	**31**
j=8	3	9	11	12	35	15	17	**20**	**23**	31
j=7	3	9	11	12	35	15	**17**	**20**	23	31
j=6	3	9	11	12	35	**15**	**17**	20	23	31
j=5	3	9	11	12	**35**	**15**	17	20	23	31
j=4	3	9	11	**12**	**15**	35	17	20	23	31
i=5; j=9	3	9	11	12	15	35	17	20	**23**	**31**
j=8	3	9	11	12	15	35	17	**20**	**23**	31
j=7	3	9	11	12	15	35	**17**	**20**	23	31
j=6	3	9	11	12	15	**35**	**17**	20	23	31
j=5	3	9	11	12	**15**	**17**	35	20	23	31
i=6; j=9	3	9	11	12	15	17	35	20	**23**	**31**
j=8	3	9	11	12	15	17	35	**20**	**23**	31
j=7	3	9	11	12	15	17	**35**	**20**	23	31
j=6	3	9	11	12	15	**17**	**20**	35	23	31
i=7; j=9	3	9	11	12	15	17	20	35	**23**	**31**
j=8	3	9	11	12	15	17	20	**35**	**23**	31
j=7	3	9	11	12	15	17	**20**	**23**	35	31
i=8; j=9	3	9	11	12	15	17	20	23	**35**	**31**
j=8	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 gelembung.

```
void BubbleSort()
{
      int i, j;
      for(i=1; i<Max-1; i++)
            for(j=Max-1; j>=i; j--)
                  if(Data[j-1] > Data[j])
                        Tukar(&Data[j-1], &Data[j]);
}
```

Program 5 Prosedur Pengurutan dengan Metode Gelembung

Dari algoritma dan prosedur diatas, jumlah perbandingan (=*C*) metode gelembung selalu konstan yaitu :

$C = (N^2 – N) / 2$

Jumlah penukaran data (=*M*) pada metode gelembung tergantung keadaan data. Jumlah penukaran minimum terjadi bila data sudah dalam keadaan urut, sebaliknya jumlah penukaran maksimum terjadi bila data dalam keadaan urut terbalik. Adapaun rumus dari jumlah penukaran pada metode gelembung adalah sebagai berikut :

$M_{min} = 0$

$M_{rata\text{-}rata} = 3(N^2 - N) / 4$

$M_{max} = 3(N^2 - N) / 2$

d. Metode Shell (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 6.4. Proses pengurutan pada tabel 6.4 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 4 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]){
```

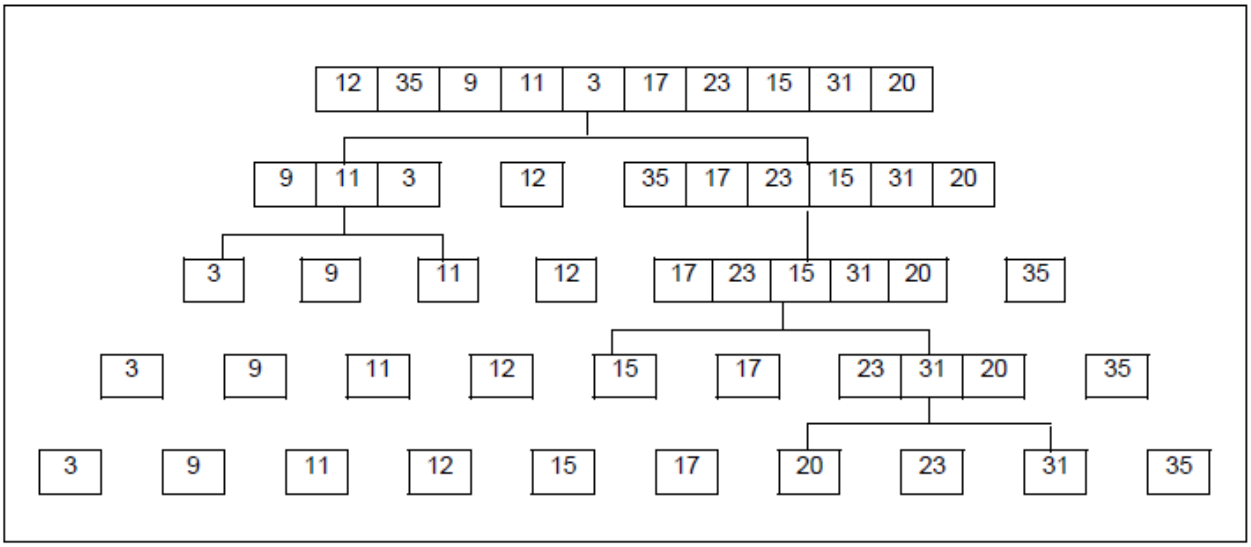
```
                Tukar(&Data[j], &Data[i]);
                Sudah = false;
            }
        }
    }
}
```

Program 6 Prosedur Pengurutan dengan Metode Shell

e. Metode Quick (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

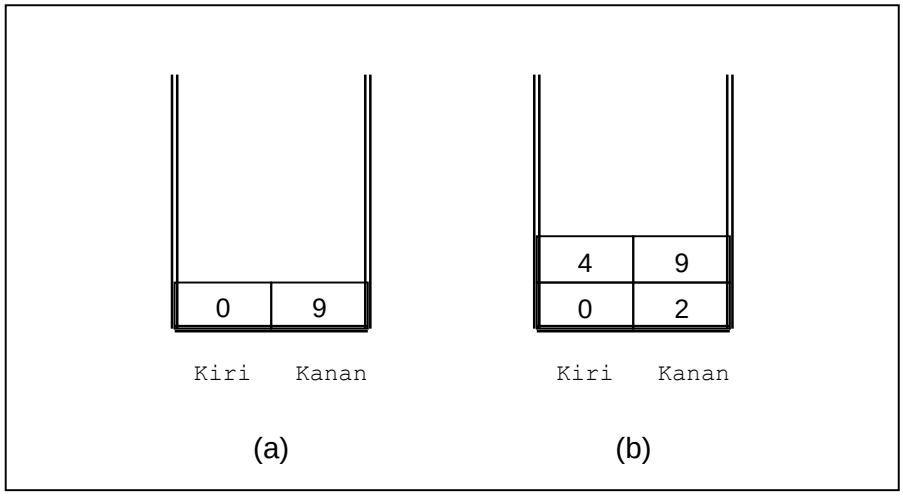
3	11	9		12		35	17	23	15	31	20

Implementasi dari metode ini secara non rekursif maupun secara rekursif dapat dilihat di sub bab di bawah ini.

f. Metode Quick Sort Non Rekursif

Implementasi secara non rekursif memerlukan dua buah tumpukan (stack) yang digunakan yang digunakan untuk menyimpan batas-batas subbagian. Pada prosedur ini menggunakan tumpukan yang bertipe record (struktur) yang terdiri dari elemen kiri (untuk mencatat batas kiri) dan kanan (untukmencatat batas kanan. Tumpukan dalama hal ini dideklarasikan sebagai array.

Dengan mengacu pada contoh diatas, sebelum metode ini dijalankan data pertama tumpukan, kiri diatur sama dengan 0 dan kanan sama dengan N-1 (pada contoh sama dengan 9), lihat Gambar 6.2a. Setelah data 12 berada pada posisi yang tepat, maka isi tumpukan berubah seperti Gambar 6.2b.



Gambar 2 Perubahan Isi Tumpukan (a) Pada Saat Mulai Iterasi
(b) Setelah Satu Elemen Ditempatkan

Algoritma quick sort non rekursif dapat dituliskan sebagai berikut :

```
1    Tumpukan[1].Kiri ← 0
2    Tumpukan[1].Kanan ← N-1
3    Selama ujung ≠ 0 kerjakan baris 4 sampai dengan 22
4    L ← Tumpukan[ujung].Kiri
5    R ← Tumpukan[ujung].Kanan
6    ujung ← ujung – 1
7    Selama (R > L) kerjakan baris sampai 8 dengan 22
8    i ← L
9    j ← R
10   x ← Data[(L + R) / 2]
11   Selama i <= j kerjakan baris 12 sampai dengan 14
12   Selama (Data[i] < x), i ← i + 1
13   Selama (x < Data[j]), j ← j – 1
14   Jika (i <= j) maka kerjakan baris 15 sampai dengan 17, jika tidak ke baris 11
15   Tukar Data[i] dengan Data[j]
16   i ← i + 1
17   j ← j –1
18   Jika (L < i) maka kerjakan baris 19 sampai dengan 21
19   ujung ← ujung + 1
20   Tumpukan[ujung].Kiri = i
21   Tumpukan[ujung].Kanan = R
22   R ← j
```

Untuk lebih memperjelas langkah-langkah algoritma quick dapat dilihat pada tabel 6.5. Proses pengurutan pada tabel 6.5 dapat dijelaskan sebagai berikut:

- Mula-mula L=0 dan R=9 dan pivot adalah pada data ke-4 yaitu 3. Kita mencari data di sebelah kiri pivot yang lebih besar daripada 3, ternyata data ke-0 yaitu 12 lebih besar daripada 3. Untuk data di sebelah kanan pivot, ternyata tidak ada data yang lebih kecil daripada 3, yang berarti 3 adalah data terkecil. Sekarang 3 harus ditukar dengan 12 seperti dilihat pada tabel 7.4.

12	35	9	11	**3**	17	23	15	31	20

⇓

3	35	9	11	**12**	17	23	15	31	20

- Langkah berikutnya membuat dua kumpulan data baru berdasarkan hasil ini. Kumpulan data pertama adalah data yang memiliki indeks 0 s/d 4-0=4 yaitu

3	35	9	11

- Kumpulan data kedua adalah data yang memiliki indeks 0 + 1 = 1 s/d 9. Kumpulan data kedua ini belum bisa ditentukan sekarang karena masih tergantung dari hasil pengurutan kumpulan data pertama.
- Kembali ke kumpulan data pertama, dicari pivot kemudian menggunakan aturan yang serupa. Pivot pada kumpulan data ini adalah data ke-2 yaitu 9. Perhatikan terdapat data yang lebih besar dari 9 di sebelah kiri yaitu 35 sehingga 35 ditukar dengan 9.

3	9	35	11

- Langkah selanjutnya membuat dua kumpulan data lagi. Kumpulan pertama mempunyai indeks 0 sampai dengan 4-1=3. Jadi kumpulan data pertama menjadi

3	9	35

- Pivot dari kumpulan data ini adalah 9. Perhatikan tidak ada data yang lebih besar dari 9 di sebelah kiri dan lebih kecil dari 9 di sebelah kanan.
- Kumpulan kedua dari indeks 0 s/d 4 adalah data ke 2 dan ke 4-1=3 yaitu 35 dan 11. Ternyata 35 lebih besar dari 11 sehingga kedua data ini ditukar.
- Kembali ke kumpulan data kedua yang memiliki indeks 1 s/d 9. Kumpulan ini dibagi menjadi dua yaitu kumpulan data berindeks 1 s/d 4 dan kumpulan data berindeks 5 s/d 9. Pivot dari data ini adalah data ke 5 yaitu 17. Kumpulan data ini dapat dituliskan sebagai berikut :

9	11	35	12	**17**	23	15	31	20

- Data yang lebih besar dari 17 di sebelah kiri adalah 35 dan data yang lebih kecil dari 17 di sebelah kanan adalah 15, jadi kedua data ini ditukar menjadi

9	11	**15**	12	17	23	**35**	31	20

- Dari hasil penukaran ini dilakukan pembagian menjadi 2 kumpulan data. Kumpulan pertama yaitu dari indeks 2 s/d 4, pivot pada data ke 3 dan terjadi penukaran data 15 dan 12 sehingga menjadi

11	12	15

- Kumpulan kedua yaitu dari indeks 4 s/d 5 tidak terjadi pertukaran data
- Kembali ke kumpulan kedua dari indeks 5 s/d 9. Pivot dari kumpulan ini adalah 35. Dicari data yang lebih besar dari 35 di sebelah kiri dan data yang lebih kecil dari 35 di sebelah kanan. Ternyata terjadi pertukaran data 35 dan 20.

- Dari hasil pertukaran ini dilakukan pembagian kumpulan data yaitu data yang mempunyai indeks 6 s/d 7 dan 8 s/9. Kumpulan data pertama terjadi pertukaran data 23 dan 20 sedangkan kumpulan data kedua tidak terjadi pertukaran data

Tabel 5 Proses Pengurutan dengan Metode Quick

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
L=0;R=9	**12**	35	9	11	**3**	17	23	15	31	20
L=0;R=4	3	**35**	**9**	11	12	17	23	15	31	20
L=1;R=3	3	9	**35**	**11**	12	17	23	15	31	20
L=1;R=9	3	9	11	**35**	12	17	23	**15**	31	20
L=2;R=4	3	9	11	**15**	**12**	17	23	35	31	20
L=5;R=9	3	9	11	12	15	17	23	**35**	31	**20**
L=6;R=7	3	9	11	12	15	17	**23**	**20**	31	35
L=8;R=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 quick dengan non rekursi

```
void QuickSortNonRekursif(int N)
{
   const M = MaxStack;
   struct tump {
      int Kiri;
      int Kanan;
   }Tumpukan[M];

   int i, j, L, R, x, ujung = 1;
   Tumpukan[1].Kiri = 0;
   Tumpukan[1].Kanan = N-1;

   while (ujung!=0){
      L = Tumpukan[ujung].Kiri;
      R = Tumpukan[ujung].Kanan;
      ujung--;
      while(R > L){
         i = L;
         j = R;
         x = Data[(L+R)/2];
```

```
            while(i <= j){
               while(Data[i] < x)
                     i++;
               while(x < Data[j])
                     j--;
               if(i <= j){
                     Tukar(&Data[i], &Data[j]);
                     i++;
                     j--;
               }
            }
            if(L < i){
               ujung++;
               Tumpukan[ujung].Kiri = i;
               Tumpukan[ujung].Kanan = R;
            }
            R = j;
         }
      }
   }
```

Program 7 Prosedur Pengurutan dengan Metode Quick Non Rekursif

g. Metode Quick Sort Rekursif

Algoritma quick Rekursif dapat dituliskan sebagai berikut :

1 x ← Data[(L + R) / 2]

2 i ← L

3 j ← R

4 Selama (i <= j) kerjakan baris 5 sampai dengan 12

5 Selama (Data[i] < x) kerjakan i ← i + 1

6 Selama (Data[j] > x) kerjakan j ← j – 1

7 Jika (i <= j) maka kerjakan baris 8 sampai dengan 10; jika tidak kerjakan baris 11

8 Tukar Data[i] dengan Data[j]

9 i ← i + 1

10 j ← j –1

11 Jika (L < j) kerjakan lagi baris 1 dengan R = j

12 Jika (i < R) kerjakan lagi baris 1 dengan L = i

Di bawah ini merupakan prosedur yang menggunakan metode quick dengan rekursi

```
void QuickSortRekursif(int L, int R)
{
     int i, j, x;
     x = data[(L+R)/2];
     i = L;
     j = R;
     while (i <= j){
          while(Data[i] < x)
               i++;
          while(Data[j] > x)
               j--;
          if(i <= j){
               Tukar(&Data[i], &Data[j]);
               i++;
               j--;
          }
     }
     if(L < j)
          QuickSortRekursif(L, j);
     if(i < R)
          QuickSortRekursif(i, R);
}
```

Program 8 Prosedur Pengurutan dengan Metode Quick Non Rekursif

3. Metode Penggabungan (Merge Sort)

Metode penggabungan biasanya digunakan pada pengurutan berkas. Prinsip dari metode penggabungan sebagai berikut : mula-mula diberikan dua kumpulan data yang sudah dalam keadaan urut. Kedua kumpulan data tersebut harus dijadikan satu table sehingga dalam keadaan urut.

Misalnya kumpulan data pertama (T1) adalah sebagai berikut :

3	11	12	23	31

Sedangkan kumpulan data kedua (T2) adalah sebagai berikut :

9	15	17	20	35

Proses penggabungan ini dapat dijelaskan sebagai berikut : mula-mula diambil data pertama dari T1 yaitu 3 dan data pertama dari T2 yaitu 9. Data ini dibandingkan, kemudian yang lebih kecil diletakkan sebagai data pertama dari hasil pengurutan, misalnya T3. Jadi T3 akan memiliki satu data yaitu 3. Data yang lebih besar yaitu 9 kemudian dibandingkan dengan data kedua dari T1, yaitu 11. Ternyata 9 lebih kecil dari 11, sehingga 9 diletakkan sebagai data kedua dari T3. Demikian seterusnya sehingga didapat hasil sebagai berikut :

3	9	11	12	15	17	20	23	31	35

Algoritma penggabungan dapat dituliskan sebagai berikut :

1 i ← 0
2 j ← 0
3 J3 ← 0
4 Kerjakan baris 5 sampai dengan 7 selama (i < J1) atau (j < J2)
5 J3 ← J3 + 1
6 Jika (T1[i] < T2[j]) maka T3[J3] ← T1[i], i ← i + 1
7 Jika (T1[i] >= T2[j]) maka T3[J3] ← T2[j], j ← j + 1
8 Jika (i > J1) maka kerjakan baris 9, jika tidak kerjakan baris 15
9 i ← j
10 Selama (i < J2) kerjakan baris 11 sampai dengan 13
11 J3 ← J3 + 1
12 T3[J3] ← T2[i]
13 i ← i + 1
14 Selesai
15 j ← i
16 Selama (j < J1) kerjakan baris 17 sampai dengan 19
17 J3 ← J3 + 1
18 T3[J3] ← T1[j]
19 j ← j + 1

Di bawah ini merupakan prosedur penggabungan dua kumpulan data yang sudah dalam keadaan urut.

```
void MergeSort(int T1[],int T2[],int J1,int J2, int T3[],int *J3)
{
      int i=0, j=0;
```

```
        int t=0;
        while ((i<J1)||(j<J2)){
              if(T1[i]<T2[j]){
                    T3[t] = T1[i];
                    i++;
              }
              else{
                    T3[t] = T2[j];
                    j++;
              }
              t++;
        }
        if(i>J1)
              for(i=j; i<J2; i++){
                    T3[t] = T2[i];
                    t++;
              }

        if(j>J2)
              for(j=i; j<J1; j++){
                    T3[t] = T1[j];
                    t++;
              }
        *J3 = t;
}
```

Program 9 Prosedur Pengurutan dengan Metode Merge

Kesimpulan

i. Metode Bubble Sort sekalipun relatif lebih mudah dilakukan tetapi tidak efisien dibandingkan dengan metode lain dalam melakukan pengurutan

ii. Metode Quick Sort dan metode Merge Sort dapat dilakukan dengan menggunakan rekursif atau tanpa rekursif

b. Latihan

i. Tuliskan kembali prosedur pengurutan dengan metode penyisipan langsung, penyisipan biner, seleksi, gelembung, Shell dan Quick agar data menjadi urut turun (*descending)*

ii. Tambahkan variable pada prosedur pengurutan untuk menghitung jumlah pembandingan dan jumlah penukaran untuk metode penyisipan langsung, penyisipan biner, seleksi, gelembung, Shell dan Quick.

iii. Buatlah fungsi menyisipkan data acak yang dibangkitkan dengan fungsi random untuk data bulat sebanyak 10000. Kemudian buatlah table yang menghitung jumlah pembandingan dan jumlah penukaran pada data acak tersebut untuk metode penyisipan langsung, penyisipan biner, seleksi, gelembung, Shell dan Quick.

Metode Pengurutan	Jumlah Pembandingan	Jumlah Penukaran
Penyisipan langsung Penyisipan biner Seleksi Gelembung Shell Quick		

iv. Tuliskan kembali prosedur pengurutan dengan metode penyisipan langsung, penyisipan biner, seleksi, gelembung, Shell dan Quick untuk data bertipe string sehingga data menjadi urut naik (a*scending)* berdasarkan nilai ASCII-nya.

v. Tuliskan program untuk mengurutkan data bertipe rekaman (record) yang mempunyai empat data yaitu :

1. Nomor Induk, bertipe bilangan bulat
2. Nama, bertipe string
3. Alamat, bertipe string
4. Golongan, bertipe char (bernilaii ‘A’ … ’Z’)

Prosedur pengurutan menerima satu parameter, yaitu bilangan bulat yang dapat bernilai 1, 2 atau 3. Apabila bernilai 1, maka data diurutkan menurut nomor induk. Apabila bernilai 2, maka data diurutkan menurut nama, dan apabila bernilai 3 maka data diurutkan menurut golongan.

vi. Tuliskan program yang bisa mengurutkan sekumpulan data bertipe rekaman berdasarkan dua kunci atau lebih. Tipe rekaman yang digunakan sama dengan latihan no. 5, yaitu Nomor Induk, Nama, Alamat dan Golongan. Data ini dapat diurutkan menurut Nama + Alamat atau Golongan + Nama. Sebagai contoh, jika data diurutkan menurut Nama + Alamat, dan terdapat dua orang dengan nama sama,

misalkan Anton dan Anton, maka dua data ini diurutkan menurut alamatnya. Jika Anton pertama beralamat di Jakarta dan Anton kedua beralamat di Bogor, maka Anton kedua harus diletakkan sebelum Anton pertama.