

# BAB VI

## SORTING

---

### 6.1 Pendahuluan

Sorting atau mengurutkan data adalah salah satu operasi yang penting dalam pengolahan data. Sorting didefinisikan sebagai suatu kegiatan untuk mengatur urutan sekelompok data sedemikian sehingga data pertama lebih kecil dari data kedua, data kedua lebih kecil dari data ketiga dan seterusnya atau data terurut dari kecil ke besar. Urutan data seperti ini disebut urut naik (*ascending*). Selain urut naik, data juga dapat diurutkan mulai dari besar ke kecil atau disebut urut turun (*decending*). Lihat gambar 6.1.

Urut naik adalah urutan data yang paling sering ditemui dalam situasi sehari-hari. Sebagai contoh: urutan nama pada buku telepon, urutan kata pada kamus, urutan nomor mahasiswa pada daftar hadir, urutan nomor buku di perpustakaan, dan sebagainya. Fakta menunjukkan bahwa data yang telah terurut akan memudahkan proses pencarian (*searching*) nantinya.

Data:	5	3	7	2	9	4	8	6
Ascending:	2	3	4	5	6	7	8	9
Descending:	9	8	7	6	5	4	3	2

Gambar 6.1 Data mula-mula dan hasil sorting

Algoritma sorting adalah algoritma yang berisi langkah-langkah untuk mengurutkan data. Ada banyak algoritma sorting yang telah dibuat orang. Berdasarkan tingkat kesulitannya, algoritma sorting dibagi menjadi 2 kelompok yaitu:

1. Sederhana (*simple*): langkah-langkahnya sederhana, mudah dipahami dan ringkas, tetapi prosesnya membutuhkan waktu yang lama. Algoritma ini cocok

untuk mengurutkan sedikit data (banyaknya data  $< 10000$ ). Algoritma yang termasuk dalam kelompok ini adalah algoritma Selection Sort, Exchange Sort dan Insertion Sort.

2. Lanjut (*advanced*): langkah-langkahnya lebih rumit dibanding algoritma yang sederhana tetapi prosesnya membutuhkan waktu yang lebih singkat. Algoritma ini biasanya digunakan untuk mengurutkan data yang banyak. Ada banyak sekali algoritma yang telah dibuat orang, contohnya Quick sort, Merge sort, Shell sort, Alpha sort, Radix sort dan sebagainya.

Dalam bab selanjutnya akan dibahas metode sorting yang sederhana dan yang lanjut. Untuk memudahkan pembahasan, ditentukan bahwa semua data yang akan diurutkan adalah bertipe integer dan disimpan dalam array linier. Data akan diurutkan secara urut naik.

## 6.2 Selection Sort

Selection sort adalah salah satu cara yang termudah untuk mengurutkan data. Sesuai namanya yang berarti pemilihan, prosesnya dimulai dengan memeriksa seluruh data sehingga dapat dipilih data yang terkecil, kemudian data tersebut ditempatkan di posisi paling awal. Selanjutnya pada data yang tersisa, dipilih lagi data yang terkecil, lalu tempatkan di posisi ke dua. Demikian seterusnya sampai seluruh data selesai diperiksa.

Algoritma Selection sort secara umum adalah :

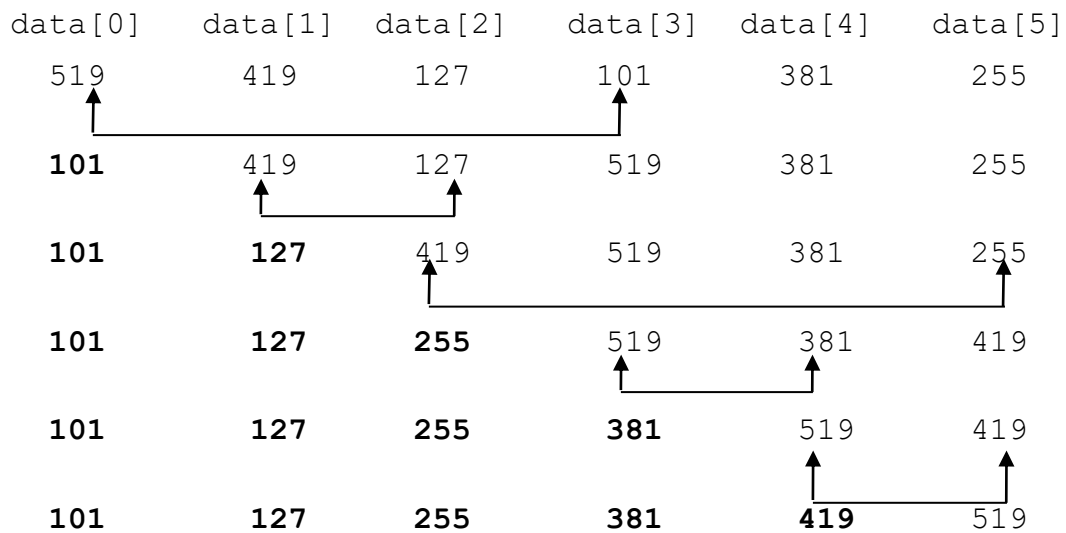
```

1. For ( I = 0 ; I < N ; I++ )
    {
        Min = I
        For ( J = I + 1 ; J <= N ; J++ )
            { Cari data terkecil, lalu tukar tempat dengan
              data pada posisi I
            }
    }
2. Halt.

```


Untuk lebih memahami algoritma di atas, dapat dilihat proses pengurutan data dengan algoritma Selection sort pada gambar 6.2 di halaman berikut. Data yang

akan diurutkan disimpan dalam array data[0..5]. Data mula-mula berada dalam kondisi tidak beraturan lalu diurutkan dengan algoritma Selection sort ini.



Gambar 5.2 Proses pengurutan data dengan algoritma selection sort

Keterangan gambar:

- Data yang dicetak tebal adalah data yang sudah pada posisinya
-  : tukar data

Proses pengurutan dengan algoritma Selection Sort ini juga dapat dilihat pada loop behavior table yang memperlihatkan hasil pencarian elemen yang terkecil dan pertukaran tempat antar elemen-elemen array data[i] dengan elemen array yang terkecil. Lihat tabel 6.1 berikut ini.

Tabel 6.1 Loop behaviour table dari algoritma Selection Sort

Loop	Data terkecil	Ditukar dengan posisi, data
I = 1	101	0, 519 ⇔ 101
I = 2	127	1, 419 ⇔ 127
I = 3	255	2, 419 ⇔ 255
I = 4	381	3, 519 ⇔ 381
I = 5	419	4, 519 ⇔ 419

Pada loop behavior table di atas dapat dilihat bahwa data dengan nilai yang besar (419 dan 519) berkali-kali bertukar tempat dengan data terkecil yang ditemukan. Algoritma Selection sort bekerja dalam waktu konstan, pemeriksaan akan tetap dilakukan sampai ke akhir array meskipun data dalam array sudah dalam keadaan terurut.

### 6.3. Exchange Sort

Algoritma Exchange sort juga termasuk algoritma yang mudah dipahami. Sesuai namanya, Exchange yang berarti pertukaran, maka dalam algoritma ini banyak terjadi pertukaran data. Cara kerja algoritma ini mulai memeriksa array dari belakang (posisi N), bandingkan elemen array pada posisi N dengan N-1, jika data pada posisi N lebih besar dari data pada posisi N-1 maka lanjutkan perbandingan antara data pada posisi N-1 dengan N-2. Jika data pada posisi N lebih kecil dari data pada posisi N-1 maka tukarkan data dan lanjutkan perbandingan antara data pada posisi N-1 dengan N-2. Demikian seterusnya sampai mencapai posisi 1. Pada saat ini data terkecil sudah berada pada posisinya. Pemeriksaan elemen array dilakukan kembali mulai dari posisi N sampai posisi 1. Dan seterusnya sampai seluruh array selesai diperiksa.

Dengan cara ini, data terkecil akan langsung menempati posisinya pada putaran pertama, tetapi tidak demikian pada data yang terbesar. Pada setiap putaran, data yang terbesar perlahan-lahan bergeser ke posisi N. Hal ini seperti gelembung sabun yang bergerak perlahan-lahan ke atas sehingga sering kali Exchange sort ini disebut juga Bubble sort.

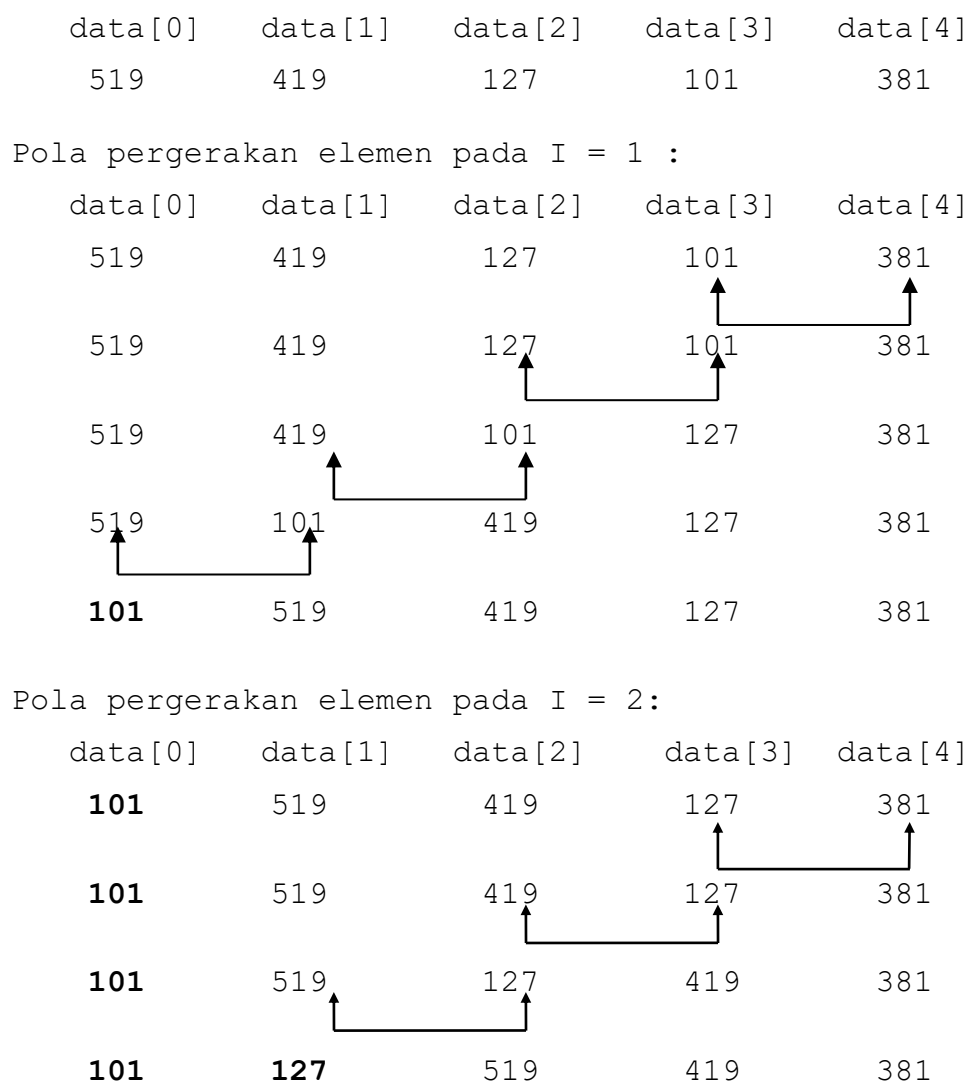
Algoritma Exchange sort secara umum adalah :

```

1. For ( I = 1 ; I <= N ; I++ )
    {
        For ( J = N ; J >= I ; J-- )
        {
            Jika data[J-1] > data [J] maka tukarkan
            posisi data
        }
    }
2. Halt.

```

Untuk lebih memahami algoritma di atas, dapat dilihat proses pengurutan data dengan algoritma Exchange sort pada gambar 6.3 di halaman berikut. Data yang akan diurutkan disimpan dalam array data[0..4]. Data mula-mula berada dalam kondisi tidak beraturan lalu diurutkan dengan algoritma Exchange sort ini. Karena banyaknya perpindahan yang terjadi maka proses digambarkan secara terperinci untuk setiap increment variabel I (loop yang di luar) dan variabel J (loop yang di dalam).



Gambar 6.3 Proses pengurutan data dengan algoritma Exchange Sort

Pola pergerakan elemen pada I = 3:

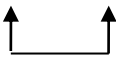
data[0]	data[1]	data[2]	data[3]	data[4]
<b>101</b>	<b>127</b>	519	419	381
<b>101</b>	<b>127</b>	519	381	419
<b>101</b>	<b>127</b>	<b>381</b>	519	419

Pola pergerakan elemen pada I = 4:

data[0]	data[1]	data[2]	data[3]	data[4]
<b>101</b>	<b>127</b>	<b>381</b>	519	419
<b>101</b>	<b>127</b>	<b>381</b>	<b>419</b>	<b>519</b>

Gambar 6.3 (lanjutan)

Keterangan gambar:

- Data yang dicetak tebal adalah data yang sudah pada posisinya
-  : bandingkan data, jika data[J-1] > data [J] maka tukar posisinya

Algoritma di atas juga tidak dapat membedakan apakah data dalam array sudah terurut atau belum sehingga pemeriksaan tetap dilakukan sampai ke akhir array meskipun data dalam array sudah terurut. Untuk mengatasi hal ini, dapat ditambahkan tanda (*flag*) yang menunjukkan apakah masih terjadi pertukaran posisi atau tidak. Jika pada suatu nilai I dengan  $I < N$  sudah tidak terjadi lagi pertukaran, maka data dalam array sudah dalam keadaan terurut sehingga pemeriksaan tidak perlu dilanjutkan lagi. Lihat fungsi `Bubble_flag(A,N)` berikut.

Void `Bubble_flag(A,N)`

Fungsi untuk mengurutkan data dengan menggunakan algoritma Bubble sort yang telah disempurnakan. Parameter A adalah array yang berisi data yang akan diurutkan. Parameter N adalah panjang array. Variabel URUT adalah flag untuk menandai apakah dalam array masih terjadi pertukaran atau tidak, berjenis logika. Variabel I dan J berjenis integer. Pertukaran elemen array dilakukan dengan fungsi `tukar(A,B)`.

```
{
    1. [inisialisasi variabel URUT]
       URUT = false
```

```

2.  [inisialisasi variabel I]
    I = 1
3.  [loop luar untuk memeriksa array]
    While I < N and (not URUT)
    {
        URUT = true
        [loop dalam untuk memeriksa tiap elemen array]
        For ( J = N ; J >= I ; J++ )
        {
            [apakah elemen sudah urut?]
            If (A[J-1] > A[J])
            { [elemen tidak urut, lakukan pertukaran]
                tukar(A[J-1], A[J])
                URUT = false
            }
        }
        [increment I]
        I = I + 1
    }
}

Void tukar(A&,B&)
Fungsi untuk melakukan pertukaran antara isi parameter A
dengan isi parameter B. Temp adalah variabel lokal.
{
    1. Temp = A
    2. A = B
    3. B = Temp
}

```

## 5.4. Insertion Sort

Algoritma Insertion sort berasal dari cara mengurutkan kartu bridge yang dilakukan oleh para pemain kartu. Sesuai namanya, algoritma ini akan mencari posisi yang sesuai untuk data yang akan diurutkan lalu menyisipkan (insert) data disitu. Jika hanya ada 1 data, maka data telah terurut. Jika ada 2 atau lebih data maka bandingkan data kedua dengan data pertama. Jika data kedua lebih kecil maka sisipkan data kedua di depan data pertama lalu lanjutkan dengan data ketiga. Jika data kedua lebih besar, lanjutkan dengan data ketiga. Dan seterusnya sampai seluruh data selesai diurutkan.

Algoritma Insertion sort secara umum adalah :

```

1.  For ( I = 1 ; I <= N ; I++ )
    {
        Jika data di posisi I < data di posisi I-1 maka
        geser data posisi I-1 ke posisi I. Jika sudah

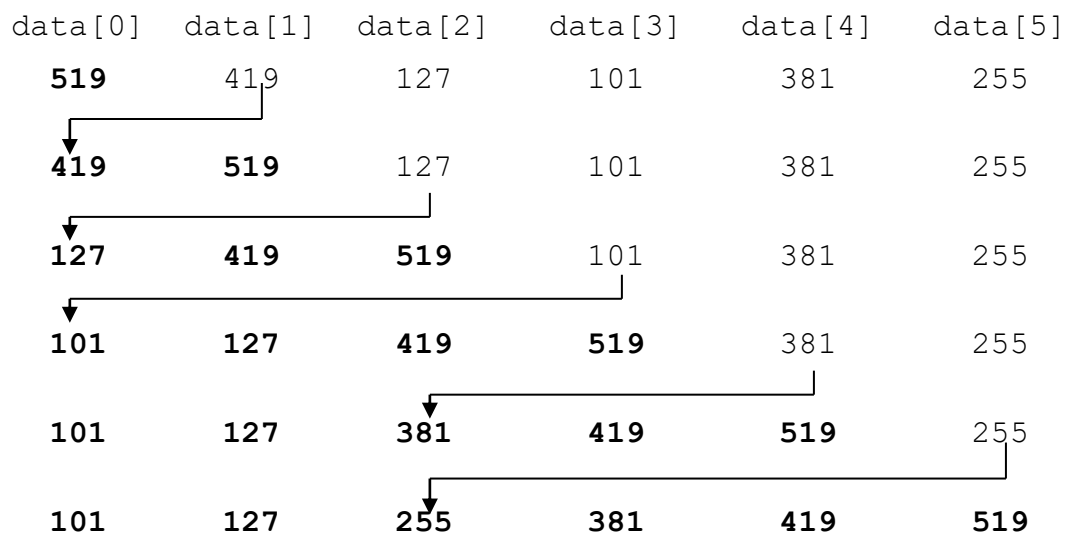
```

sesuai sisipkan data di posisi tersebut. Jika belum, lanjutkan perbandingan dengan data di depannya (I-1 dengan I-2, dan seterusnya) sampai ditemui posisi data yang sesuai lalu sisipkan data di posisi tersebut.

}

2. Halt.

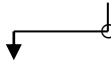
Untuk jelasnya dapat dilihat pada proses pengurutan data dengan algoritma Insertion sort pada gambar 6.4 berikut ini. Data yang akan diurutkan disimpan dalam array data[0..5]. Data mula-mula berada dalam kondisi tidak beraturan lalu diurutkan dengan algoritma Insertion sort ini. Data dalam array akan diperiksa satu persatu mulai dari data pertama. Pada langkah pertama, data di posisi 1 sudah dalam keadaan terurut karena hanya ada 1 data yang diperiksa. Pada langkah-langkah berikutnya akan dicari posisi data yang sesuai dengan mengeser data yang tidak pada tempatnya.



Gambar 6.4 Proses pengurutan data dengan algoritma Insertion sort

Keterangan gambar:

- Data yang dicetak tebal adalah data yang sudah pada posisinya

 : jika data  $[i+1] < \text{data}[i]$ , geser data  $[i]$  ke  $[i+1]$  lalu sisipkan data  $[i+1]$  ke posisi  $[i]$ .



Dibanding dengan kedua algoritma yang telah dibahas, algoritma Insertion sort adalah algoritma yang paling efisien karena pergeseran hanya dilakukan jika diperlukan. Bandingkan dengan algoritma Selection sort yang terus mencari data terkecil dan Exchange sort yang banyak melakukan pertukaran ketika mencari posisi data yang sesuai.

Pada gambar 6.4 di atas terlihat bahwa terjadi banyak pergeseran untuk memberi tempat bagi data yang harus menempati posisi yang seharusnya. Hal ini akan memerlukan waktu yang lama jika data yang diurutkan sangat banyak, untuk mengurangi pergeseran ini dapat digunakan struktur data linked list sebagai tempat menyimpan data.

### **Latihan soal subbab 6.4**

1. Diketahui data yang disimpan dalam array A[0..8] berikut ini:  
 $A[7, 5, 9, 3, 6, 1, 8, 2, 4]$   
 Urutkan data dalam array A dengan algoritma berikut. Gambarkan prosesnya dan hitung banyaknya perpindahan data atau pertukaran data yang terjadi.
  - 1.a. Selection sort
  - 1.b. Exchange sort
  - 1.c. Bubble flag sort
  - 1.d. Insertion sort
- 2.a. Lengkapi algoritma Selection sort, Exchange Sort dan Insertion Sort dengan fungsi dan algoritma utama
- 2.b. Buat program dalam bahasa C++ untuk algoritma-algoritma sorting pada nomor 1 di atas lalu jalankan dan catat waktu yang dibutuhkan untuk mengurutkan 1000, 2000, 4000, 8000, dan 16000 data acak integer. Catat pula spesifikasi komputer yang anda gunakan untuk menjalankan program ini.

## **6.5. Mergesort**

Algoritma Mergesort merupakan salah satu algoritma sorting yang pertama yang digunakan oleh komputer. Algoritma ini ditulis oleh John Von Neuman pada

tahun 1945 untuk “komputer” EDVAC (Electronic Discrete Variable Automatic Computer). Sesuai namanya, merge yang artinya menggabungkan, algoritma ini akan menggabungkan 2 deretan data ( $A[n]$  dan  $B[m]$ ) yang masing-masing sudah terurut menjadi 1 deretan data ( $C[m+n]$ ) yang juga terurut. Caranya adalah dengan membandingkan elemen array A dan B satu persatu. Jika elemen array A lebih kecil maka salin ke array C, lalu lanjutkan dengan elemen berikutnya. Jika elemen array B lebih kecil maka salin ke array C, lalu lanjutkan dengan elemen berikutnya. Perbandingan ini dilakukan sampai seluruh elemen array A dan B selesai dibandingkan dan disalin ke array C.

Sebagai contoh ada 2 deret data yang disimpan dalam array A dan array B. Kedua deret data ini akan digabungkan menjadi satu deret data yang disimpan dalam array C. Hasil penggabungan adalah deretan data yang juga terurut. Lihat gambar 6.5 berikut.

```

Array A :  10  13  24  26
Array B :  12  15  27  38
Array C :  10  12  13  15  24  26  27  38

```

Gambar 6.5 Contoh penggabungan 2 deretan data

Void SIMPLE\_MERGE(A, B, C, N, M)  
 Fungsi untuk menggabungkan 2 deret data,  $A[N]$  dan  $B[M]$ , yang sudah terurut menjadi satu deret data,  $C[M+N]$  yang juga terurut. I, J, dan K adalah variabel integer yang berfungsi sebagai counter untuk masing-masing array.

```

{
  1.  [inisialisasi counter]
      I = 1
      J = 1
      K = 1
  2.  [menelusuri array A dan B, sambil menyalin elemen
      array A dan B ke array C]
      While ((I <= N) && (J <= M))
      {
  2.a  [jika elemen array A lebih kecil dari B]
        If ( A[I] <= B[J] )
        { C[K] = A[I] [ salin elemen array A ]
          K = K + 1      [ increment counter ]
          I = I + 1    }
  2.b  [jika elemen array A lebih besar dari B]
        else
        { C[K] = B[J] [ salin elemen array B ]
          K = K + 1      [ increment counter ]

```

```

        J = J + 1    }
    }
3.  [jika array A belum selesai ditelusuri]
    While ( I <= N )
    { C[K] = A[I]      [ salin sisa elemen array A ]
      K = K + 1        [ increment counter ]
      I = I + 1
    }
4.  [jika array B belum selesai ditelusuri]
    While ( J <= M )
    { C[K] = B[J]      [ salin sisa elemen array B ]
      K = K + 1        [ increment counter ]
      J = J + 1
    }
}
    
```

Untuk jelasnya, lihat tabel 6.2 yang menunjukkan hasil penelusuran fungsi simple merge dengan digunakan contoh pada gambar 6.5 di atas.

Tabel 6.2 Trace tabel dari fungsi Simple\_Merge

Langkah	I	J	K	Loop	C[1]	C[2]	C[3]	C[4]	C[5]	C[6]	C[7]	C[8]
1	1	1	1		?	?	?	?	?	?	?	?
2	1	1	1	T	?	?	?	?	?	?	?	?
2.a	2	1	2		10	?	?	?	?	?	?	?
2	2	1	2	T	10	?	?	?	?	?	?	?
2.b	2	2	3		10	12	?	?	?	?	?	?
2	2	2	3	T	10	12	?	?	?	?	?	?
2.a	3	2	4		10	12	13	?	?	?	?	?
2	3	2	4	T	10	12	13	?	?	?	?	?
2b	3	3	5		10	12	13	15	?	?	?	?
2	3	3	5	T	10	12	13	15	?	?	?	?
2a	4	3	6		10	12	13	15	24	?	?	?
2	4	3	6	T	10	12	13	15	24	?	?	?
2.a	5	3	7		10	12	13	15	24	26	?	?
2	5	3	7	F	10	12	13	15	24	26	?	?
3	5	3	7	F	10	12	13	15	24	26	?	?
4	5	4	8	T	10	12	13	15	24	26	27	?
4	5	5	9	F	10	12	13	15	24	26	27	38
5	5	5	9		10	12	13	15	24	26	27	38

Algoritma Simple Merge di atas dapat dimodifikasi sehingga dapat mengurutkan data yang berada dalam sebuah array. Caranya mula-mula array dibagi menjadi 2 subarray yang panjangnya sama. Lalu subarray ini dibagi menjadi 2 lagi secara rekursif sampai akhirnya tiap subarray mempunyai 1 elemen. Lalu gabungkan subarray-subarray tersebut dengan menggunakan algoritma simple merge secara rekursif sehingga kembali menjadi 2 subarray yang masing-masing sudah terurut. Penggabungan 2 subarray yang terakhir ini akan menghasilkan sebuah array yang sudah terurut.

Void MergeSort(Key,A,N)

Fungsi untuk mengurutkan data dengan algoritma Mergesort secara rekursif. Algoritma akan membagi array menjadi 2 subarray sampai banyaknya elemen dalam tiap subarray = 1, lalu memanggil fungsi Merge untuk menggabungkan 2 subarray menjadi sebuah array. Key adalah array yang berisi data yang akan diurutkan. A adalah indeks awal. N adalah banyaknya elemen array.

```
{
    1. [jika banyaknya elemen > 1 maka array dibagi menjadi 2]
        If (N > 1)
        {
            1.a [dibagi menjadi 2 subarray]
                m = (N+A)/2
            1.b [mergesort tiap subarray]
                MergeSort(Key, A, m)
            1.c MergeSort(Key, m+1, N)
            1.d [gabungkan kedua subarray]
                Merge(Key, A, m, N)
        }
}
```

Void Merge(Arr, i, m,j)

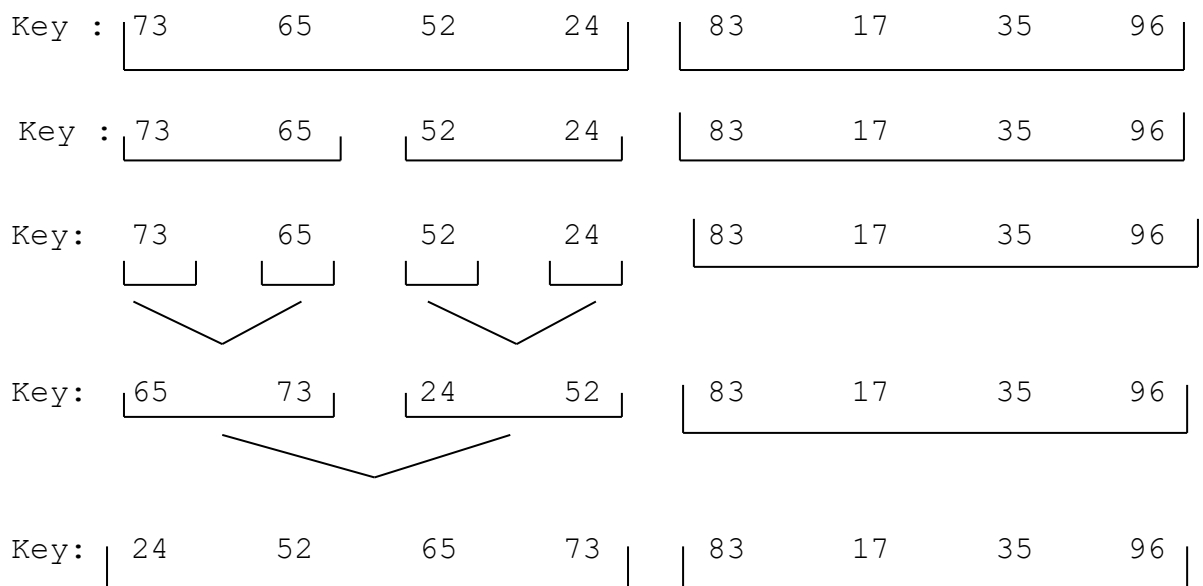
Fungsi untuk menggabungkan isi array Arr[i], ..., A[m] dengan array Arr[m+1], ..., Arr[j] menjadi sebuah array Brr[i], ..., Brr[j].

```
{
    1. [inisialisasi untuk index array Arr[i], ..., Arr[m]]
        p = i
    2. [inisialisasi untuk index array Arr[m+1], ..., Arr[j]]
        q = m + 1
    3. [inisialisasi untuk index array Brr[i], ..., Brr[j],
        tempat menyimpan hasil penggabungan]
        r = i
    4. While (p <= m && q <= j)
        {
            4.a [salin nilai yang lebih kecil ke array Brr]
                If (Arr[p] <= Arr[q])
```

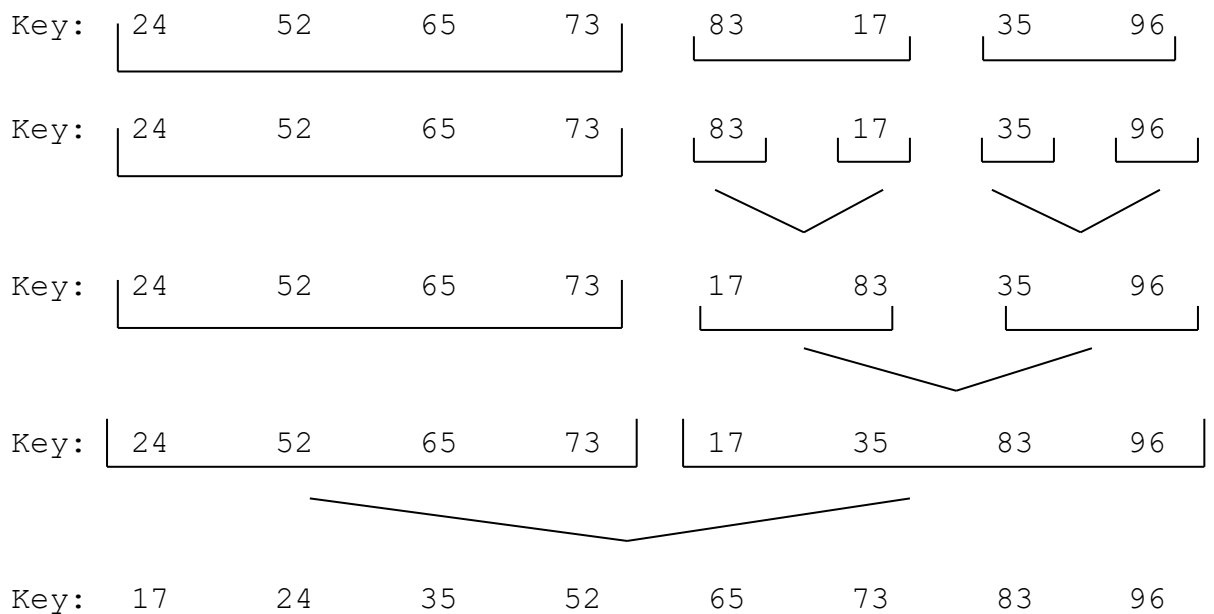
```

        {
            Brr[r] = Arr[p]
            p++
        }
    4.b else
        {
            Brr[r] = Arr[q]
            q++
        }
    4.c r++
    5. [Salin sisa array Arr[i], ..., Arr[m] ke array Brr]
    While (p <= m)
    {
        Brr[r] = Arr[p]
        r++
        p++
    }
    6. [Salin sisa array Arr[m+1], ..., Arr[j] ke array Brr]
    While (q <= j)
    {
        Brr[r] = Arr[q]
        r++
        q++
    }
    7. [salin isi array Brr ke Arr]
    For (r = 1 ; r <= j, r ++ )
    {
        Arr[r] = Brr[r]
    }
}
    
```

Proses pengurutan dengan algoritma Merge sort dapat dilihat pada gambar 6.6 di bawah ini. Array Key adalah array mula-mula berisi 8 data acak. Temp adalah array sementara yang digunakan untuk menyimpan hasil penggabungan subarray.



Gambar 6.6 Proses pengurutan dengan menggunakan algoritma Mergesort



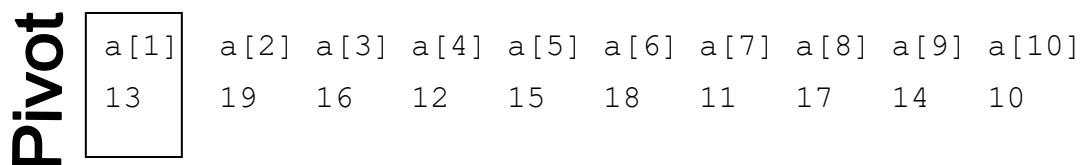
Gambar 6.6 (lanjutan)

## 6.5. Quicksort

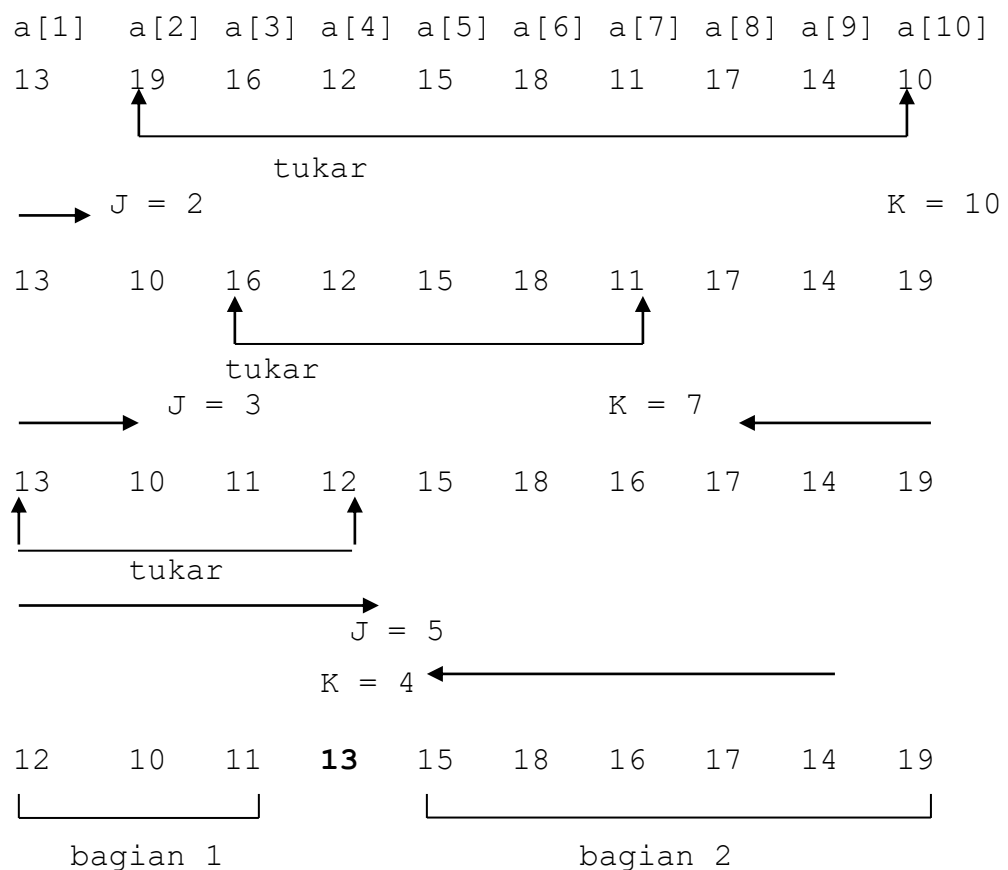
Algoritma Quicksort ditulis oleh C.A.R. Hoare pada tahun 1962. Algoritma ini merupakan pengembangan dari algoritma Exchange sort. Dalam proses pengurutan data, deretan data dipilah-pilah menjadi deretan data yang lebih pendek untuk diurutkan, karena itu algoritma ini disebut juga Partition-Exchange sort. Pemilahan data dilakukan dengan menggunakan data pemisah (yang disebut pivot) sedemikian sehingga deretan data yang berada di sebelah kiri pivot ini nilainya lebih kecil dan deretan data di sebelah kanan pivot bernilai lebih besar. Setelah itu, pada kedua bagian (*partition*) tadi dilakukan hal yang sama dengan menggunakan pivot lain. Proses pemilahan dilakukan secara rekursif sampai tiap bagian hanya terdiri dari 0 sampai 1 data.

Sebagai contoh dapat dilihat proses pengurutan data dengan algoritma Quick sort pada gambar 6.7 dan gambar 6.8 . Pada gambar 6.7, array A berisi 10 data acak yang akan diurutkan. Sebagai pivot diambil data pertama, A[1], batas kiri, L = 1, dan batas kanan R = 10. Untuk mencari elemen array yang nilainya lebih besar

dari pivot, digunakan variabel J yang menelusuri array dari kiri ke kanan. Untuk mencari elemen yang nilainya lebih kecil dari pivot digunakan variabel K yang menelusuri array dari kanan ke kiri. Jika  $J \leq K$  maka tukarkan  $A[J]$  dengan  $A[K]$  lalu lanjutkan penelusuran. Jika  $J > K$  maka tukarkan  $A[K]$  dengan  $A[L]$  dan penelusuran selesai. Pada saat ini array A sudah terbagi menjadi 2 bagian, yaitu dari  $A[L]$  sampai  $A[K-1]$  yang nilai elemennya lebih kecil dari pivot dan dari  $A[K+1]$  sampai  $A[R]$  yang nilai elemennya lebih besar dari pivot. Selanjutnya untuk kedua bagian ini dilakukan hal yang sama sehingga tiap bagian hanya terdiri dari 0 sampai 1 elemen.



Gambar 6.7 Array A mula-mula



Gambar 6.8 Proses pengurutan data dengan algoritma Quick sort

Dalam gambar 6.8, dapat dilihat bahwa posisi data yang bernilai 13 di A[4] sudah pada posisi yang sebenarnya karena seluruh data di sebelah kirinya bernilai lebih kecil dan seluruh data di kanannya bernilai lebih besar. Secara rekursif dilakukan hal yang sama untuk subarray sebelah kiri dengan L = 1 dan R = 3. Setelah selesai lakukan hal yang sama untuk subarray sebelah kanan dengan L = 5 dan R = 10. Proses pengurutan data pada array A berikutnya dapat dilihat pada gambar 4.9 di halaman berikut.

Pada Quicksort, pemilihan pivot sangat penting karena akan menentukan bagaimana array dibagi menjadi 2 bagian. Yang diharapkan adalah pivot selalu dapat membagi array menjadi 2 bagian yang sama panjangnya. Contoh yang ditampilkan pada gambar 5.9 merupakan contoh dari pemilihan pivot yang kurang baik karena array tidak terbagi menjadi 2 bagian yang sama panjangnya. Bahkan dapat terjadi array tidak terbagi sama sekali karena pivot merupakan data terkecil. Hal ini menyebabkan performansi algoritma Quicksort menjadi buruk.

a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]	a[10]
{ 12	10	11 }	<b>13</b>	{ 15	18	16	17	14	19 }
{ 11	10 }	<b>12</b>	<b>13</b>	{ 15	18	16	17	14	19 }
{ 10 }	<b>11</b>	<b>12</b>	<b>13</b>	{ 15	18	16	17	14	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	{ 15	18	16	17	14	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	{ 14 }	<b>15</b>	{ 16	17	18	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	{ 16	17	18	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>	{ 17	18	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>	<b>17</b>	{ 18	19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>	<b>17</b>	<b>18</b>	{ 19 }
<b>10</b>	<b>11</b>	<b>12</b>	<b>13</b>	<b>14</b>	<b>15</b>	<b>16</b>	<b>17</b>	<b>18</b>	<b>19</b>

Gambar 6.9 Proses pengurutan data dengan algoritma Quick Sort

Keterangan gambar:

- cetak miring: pivot
- cetak tebal : data sudah pada posisi sebenarnya (terurut)

Void QUICKSORT(A, L, R)

Fungsi untuk mengurutkan data dengan algoritma Quicksort. A adalah array tempat menyimpan data acak. L batas bawah array. R batas atas array. J dan K variabel untuk menelusuri



array. Variabel J untuk mencari elemen yang nilainya lebih besar dari pivot. Variabel K untuk mencari elemen yang nilainya lebih kecil dari pivot. Sebagai pivot dipilih elemen yang pertama. Semua variabel dan parameter berjenis integer.

```
{
1.  [ apakah sudah selesai? ]
    If ( L < R )
    { [menelusuri array untuk mencari elemen yang lebih
      kecil dan lebih besar dari pivot]
      J = L+1          [ inisialisasi variabel ]
      K = R
      [loop untuk menelusuri array]
      While ( J < K )
      { While ( A[J] < A[L] )      [ mencari elemen yang ]
        { J = J + 1              [ lebih kecil dari pivot]
        }
        While ( A[K] > A[L] )    [ mencari elemen yang ]
        { K = K - 1              [ lebih besar dari pivot]
        }
        If ( J < K )              [ J dan K belum bersilangan]
        { TUKAR(A[J], A[K])      [ tukarkan elemen ]
        }
      }
      [J dan K sudah bersilangan]
      TUKAR(A[L], A[K])          [ tukarkan pivot dengan A[K] ]
      QUICKSORT(A, L, K-1)       [ panggil quicksort untuk ]
                                [ mengurutkan data di kiri pivot ]
      QUICKSORT(A, K+1, R)       [ panggil quicksort untuk ]
                                [ mengurutkan data di kanan pivot]
    }
}
```

Ada beberapa cara dalam menentukan pivot yaitu dengan menggunakan :

- a. data yang pertama (lihat contoh yang sudah dibahas)
- b. data yang di tengah:  $n \div 2$  ; n adalah panjang array
- c. data median dari A[L], A[(n div 2)], dan A[R]

Dari ketiga cara di atas, cara yang ketiga adalah cara yang paling besar kemungkinannya untuk mendapatkan pivot yang baik, tetapi perlu dibuat fungsi khusus untuk menentukan pivot ini.

## 6.7. Shellsort

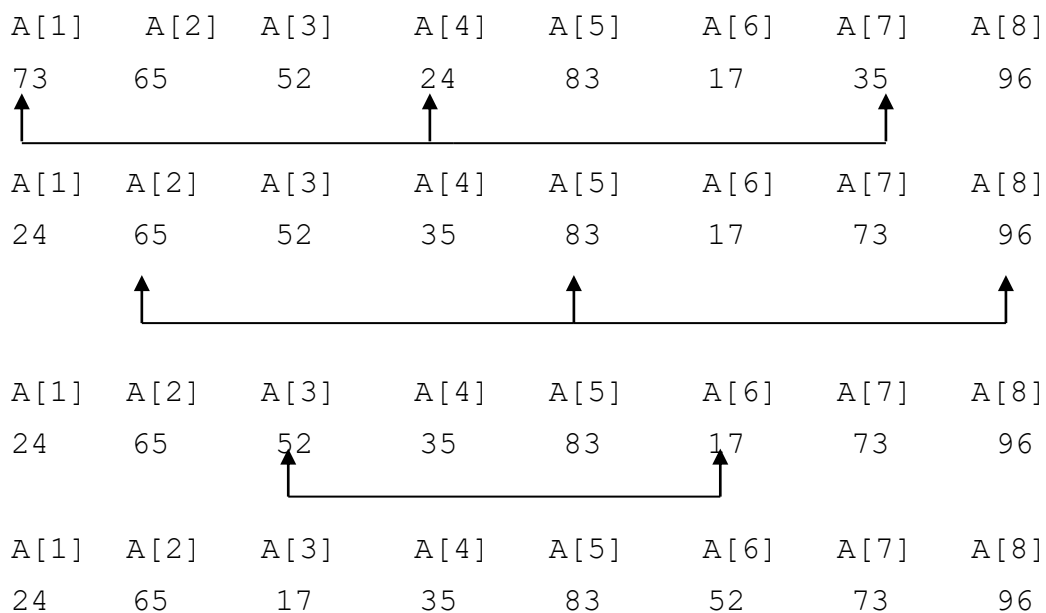
Shellsort ditulis oleh Donald L. Shell pada tahun 1959. Idenya adalah untuk memperbaiki performasi algoritma Insertion sort. dengan cara memperpendek

“jarak” penyisipan elemen. Shellsort menggunakan nilai delta (jarak antar elemen yang diperiksa) untuk membagi data yang akan diurutkan menjadi beberapa bagian. Setiap bagian tersebut kemudian diurutkan dengan menggunakan insertion sort. Pengurutan diulangi dengan delta yang lebih kecil karena itu Shellsort disebut juga “a diminishing sort” karena nilai delta akan semakin mengecil hingga mencapai nilai akhir yaitu 1.

Ada beberapa cara yang diusulkan oleh para ilmuwan komputer untuk menentukan nilai delta yaitu :

1. Mark A. Weiss :  $2^{k-1}, \dots, 15, 7, 3, 1$
2. Donald E. Knuth :  $h_0 = 1, h_{s+1} = 3h_s + 1$

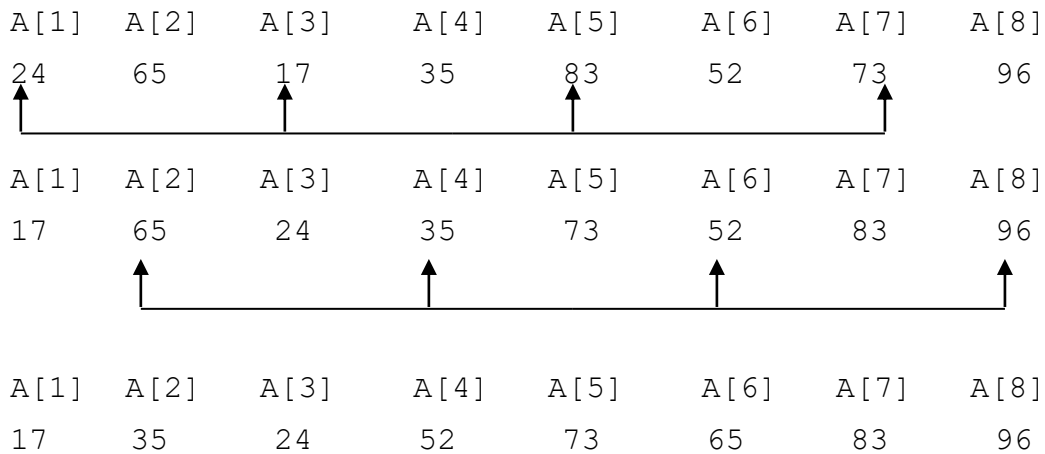
Proses pengurutan data dengan algoritma Shellsort ini dapat dilihat pada gambar 6.10. Sebagai contoh akan diurutkan 8 data acak ( $n = 8$ ) yang disimpan dalam array A. Sebagai delta, rumus dari Donald E. Knuth dimodifikasi untuk menentukan delta mula-mula, yaitu **delta = 1 + n/3** atau  $\text{delta} = 1 + 8/3 = 3$ . Dengan demikian insertion sort akan dilakukan untuk data pada A[1], A[4] dan A[7], lalu pada A[2], A[5] dan A[8], dan terakhir pada A[3] dan A[6], lihat gambar 6.10.a.



Gambar 6.10.a Pengurutan data dengan algoritma Shellsort dengan  $\text{delta} = 3$

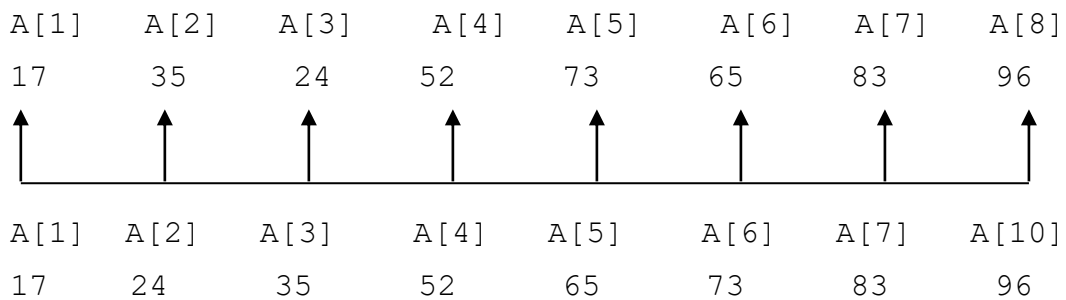
Untuk proses selanjutnya dihitung delta yang baru, yaitu  $\text{delta} = 1 + 3/3 =$

2. Dengan demikian insertion sort akan dilakukan untuk A[1], A[3], A[5], dan A[7] lalu untuk A[2], A[4], A[6], dan A[8], lihat gambar 6.10.b.



Gambar 6.10.b Pengurutan data dengan algoritma Shellsort dengan  $\text{delta} = 2$

Selanjutnya, dihitung delta yang baru, yaitu  $\text{delta} = 1 + 2/3 = 1$ , lihat gambar 6.10.c. Proses ini merupakan proses pengurutan yang terakhir dalam Shell sort.



Gambar 6.10.c Pengurutan data dengan algoritma Shellsort dengan  $\text{delta} = 1$

Void ShellSort(A,N)

Pengurutan data dengan algoritma Shellsort. A adalah array tempat data acak. N adalah panjang array. Untuk menentukan delta digunakan rumus yang diusulkan oleh Donald L. Knuth. Pengurutan dilakukan oleh fungsi DeltaInsertionSort. Semua variabel dan parameter berjenis integer.

```
{
1.  [inisialisasi variabel]
    DELTA = N
2.  While ( DELTA > 1 )
    {
2.a  [menentukan nilai delta]
```

```

        DELTA = 1 + DELTA/3
2.b    [membuat loop untuk memeriksa elemen array]
        For ( I = 1 ; I <= DELTA ; i++ )
        {
            [panggil fungsi DeltaInsertionSort]
            DeltaInsertionSort (A,I,DELTA,N)
        }
    }
}
Void DeltaInsertionSort (A,I,DELTA,N)
Fungsi untuk mengurutkan data secara insertion sort dengan
menggunakan delta.
{
1.    [inisialisasi variabel]
    j = I + DELTA
2.    [membuat loop untuk memeriksa array]
    While ( j < N )
    {
2.a    [data yang diperiksa]
        KEYTOINSERT = A[j]
2.b    [inisialisasi variabel]
        k = j
        NOTDONE = TRUE
2.c    [loop untuk mencari posisi key yang sebenarnya]
        While ( NOTDONE )
        { If ( A[k-DELTA] ≤ KEYTOINSERT )
            { NOTDONE = FALSE }
            else
            { A[k] = A[k-DELTA]
              k = k - DELTA
              If ( k == I )
              { NOTDONE = FALSE }
            }
        }
2.d.    [sisipkan key di tempat seharusnya]
        A[k] = KEYTOINSERT
2.e.    [periksa key berikutnya]
        j = j + DELTA
    }
}
    
```

## 6.8. Radixsort

Berbeda dengan algoritma-algoritma di atas, Radix Sort adalah algoritma pengurutan data yang tidak memerlukan perbandingan dalam prosesnya (Non-Comparison Sort), tetapi algoritma ini memerlukan ruang tambahan yang cukup besar dalam prosesnya dan hanya dapat diterapkan pada data numerik (bilangan bulat atau integer).

Radix dapat diartikan sebagai posisi dalam angka, karena metode ini pertama kali akan membagi data set ke sub-sub data set sesuai dengan harga radix-nya, mulai dari radix yang paling tidak signifikan (least significant digit - lsd), kemudian menggabungkan subset-subset tersebut menjadi satu set kembali (hanya mengkonkatenasi) untuk dilakukan pembagian berikutnya berdasarkan digit yang lebih signifikan. Pada sistem desimal, radix adalah digit dalam angka desimal.

Dalam pengertian yang lain, algoritma Radix sort adalah algoritma yang mengurutkan data berdasarkan angka integer dengan mengelompokkannya berdasarkan digit individu yang mempunyai posisi dan nilai yang sama dalam satu tempat (bucket)

Sebagai contoh diketahui data berikut ini:

170, 45, 75, 90, 802, 2, 24, 66

Data di atas terdiri dari maksimum 3 digit, maka perlu dilakukan 3 kali pengelompokan dan penggabungan untuk mendapatkan data yang sudah terurut. Pengelompokan yang pertama dilakukan terhadap digit yang paling tidak signifikan (lsd), yaitu digit yang ke 3. Untuk itu disiapkan 10 subset untuk menyimpan (sementara) data yang mempunyai lsd yang sama. Hasilnya adalah:

1. Subset digit 0 berukuran 2 bucket: 170, 090
2. Subset digit 1 berukuran 0 bucket
3. Subset digit 2 berukuran 2 bucket: 802, 002
4. Subset digit 3 berukuran 0 bucket
5. Subset digit 4 berukuran 1 bucket: 024
6. Subset digit 5 berukuran 2 bucket: 045, 075
7. Subset digit 6 berukuran 1 bucket: 066
8. Subset digit 7 berukuran 0 bucket
9. Subset digit 8 berukuran 0 bucket
10. Subset digit 9 berukuran 0 bucket

Hasil dari pengelompokan terhadap lsd di atas, digabungkan kembali dengan menyambung (konkatenasi) mulai dari subset digit 0. Hasil penggabungannya:

170, 090, 802, 002, 024, 045, 075, 066

Pengelompokan yang kedua dilakukan terhadap digit yang lebih signifikan, yaitu digit yang ke 2. Untuk 10 subset yang sama seperti di atas, data dikelompokkan berdasarkan digit yang ke 2. Hasilnya adalah:

1. Subset digit 0 berukuran 2 bucket: 802, 002
2. Subset digit 1 berukuran 0 bucket
3. Subset digit 2 berukuran 1 bucket: 024
4. Subset digit 3 berukuran 0 bucket
5. Subset digit 4 berukuran 1 bucket: 045
6. Subset digit 5 berukuran 0 bucket
7. Subset digit 6 berukuran 1 bucket: 066
8. Subset digit 7 berukuran 2 bucket: 170, 075
9. Subset digit 8 berukuran 0 bucket
10. Subset digit 9 berukuran 1 bucket: 090

Hasil dari pengelompokan terhadap digit kedua di atas, digabungkan kembali dengan menyambung (konkatenasi) mulai dari subset digit 0. Hasil penggabungannya:

802, 002, 024, 045, 066, 170, 075, 090

Pengelompokan yang ketiga dan terakhir adalah pengelompokan berdasarkan digit yang paling signifikan (most significant digit – msd) sehingga akan menghasilkan:

1. Subset digit 0 berukuran 6 bucket: 002, 024, 045, 066, 075, 090
2. Subset digit 1 berukuran 1 bucket: 170
3. Subset digit 2 berukuran 0 bucket
4. Subset digit 3 berukuran 0 bucket
5. Subset digit 4 berukuran 0 bucket
6. Subset digit 5 berukuran 0 bucket
7. Subset digit 6 berukuran 0 bucket
8. Subset digit 7 berukuran 0 bucket
9. Subset digit 8 berukuran 1 bucket: 802
10. Subset digit 9 berukuran 0 bucket

Hasil dari pengelompokan terhadap msd di atas, digabungkan kembali dengan menyambung (konkatenasi) mulai dari subset digit 0. Hasil penggabungannya:

002, 024, 045, 066, 075, 090, 170, 802

Dalam contoh di atas, sebagai subset dapat digunakan array yang masing-masing panjangnya sebanyak data yang akan diurutkan, sehingga untuk kasus di atas, diperlukan tambahan ruang memori sebesar 10 x banyaknya data. Radix sort juga dapat diterapkan untuk data octal, hexadecimal atau karakter. Waktu yang dibutuhkan untuk mengurutkan data cukup singkat, tetapi memerlukan tambahan ruang yang cukup banyak ketika melakukan proses pengurutan.

### **LATIHAN SOAL BAB 5.7**

1. Diketahui data yang disimpan dalam array A[0..15] berikut ini:  
A[17, 3, 15, 10, 14, 9, 12, 4, 5, 2, 16, 7, 19, 11, 14, 8]  
Urutkan data dalam array A dengan algoritma berikut ( gambarkan prosesnya)
  - 1.a. Merge sort
  - 1.b. Quicksort
  - 1.c Shellsort
  - 1.d Radix sort
2. Buat program dalam bahasa C++ untuk algoritma-algoritma sorting pada nomor 1 di atas lalu jalankan dan catat waktu yang dibutuhkan untuk mengurutkan 4000, 8000, 12000, 16000 dan 32000 data acak integer. Catat pula spesifikasi komputer yang anda gunakan untuk menjalankan program ini.