

## 5 Sorting

*“The worse is better”*

*-Me*

**B**ab ini membahas tentang metode pengurutan (*sorting*) data. Pembahasan berfokus pada 3 metode pengurutan yaitu *bubble*, *insertion*, dan *selection*. Tiap metode dibahas menggunakan contoh dan disertai dengan ilustrasi visual untuk menggambarkan tiap prosesnya. Di akhir pembahasan dilanjutkan dengan contoh kasus berupa perbandingan kecepatan di antara ketiga metode. Selain itu juga diperkenalkan secara singkat tentang fungsi dalam *standard library* C++ untuk pengurutan.

## 5.1 Pengertian

Sorting atau pengurutan diartikan sebagai proses penyusunan kembali sekumpulan obyek ke dalam urutan tertentu. Tujuan pengurutan untuk mendapatkan kemudahan dalam pencarian anggota dari suatu himpunan disamping dapat mempercepat mengetahui data terbesar dan data terkecil, misalnya untuk mengetahui perolehan nilai tertinggi dan nilai terendah dari hasil ujian. Contoh obyek terurutkan seperti daftar isi, daftar pustaka dan lain-lain.

Dalam kehidupan sehari – hari juga sering ditemukan aktivitas pengurutan seperti pengurutan susunan kartu remi, pengurutan susunan berkas dalam filing cabinet, dan lain sebagainya.

Secara umum proses yang terjadi pada pengurutan adalah sebagai berikut:

1. Perbandingan data
2. Pertukaran data

## 5.2 Metode

Ada beberapa metode sorting yang dapat digunakan. Masing – masing metode memiliki kelebihan dan kekurangan. Beberapa metode tersebut di antaranya adalah sebagai berikut.

1. Selection Sort
2. Bubble Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Heap Sort

Di sini ini hanya akan membahas beberapa metode sorting yang paling sederhana yaitu selection sort, bubble sort, dan insertion sort.

### 5.2.1 Selection Sort

Metode selection mengadopsi dari proses pengurutan dalam dunia nyata, salah satu contohnya adalah pengurutan kartu. Prosesnya dilakukan dengan menyeleksi data terbesar (atau terkecil) kemudian meletakkannya di ujung sebagai data yang sudah terurut. Proses penyeleksian akan terus dilakukan terhadap sisa data yang belum terurut hingga semua data sudah terurut.


Sebagai ilustrasi diberikan sebuah array bernama MyArray dengan data {5, 9, 7, 3, 6, 4}. Dengan metode selection maka pengurutan naik (ascending) dapat diselesaikan menggunakan langkah – langkah seperti berikut ini.

1. Kondisi awal

0	1	2	3	4	5
5	9	7	3	6	4

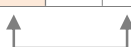
2. Langkah ke-1, ditemukan data terbesar 9 pada posisi index 1, ditukar dengan data pada index 5.

0	1	2	3	4	5
5	9	7	3	6	4




3. Langkah ke-2, data pada index 5 sudah terurut. Dari yang tersisa ditemukan data terbesar yaitu 7 pada posisi index 2 kemudian ditukar dengan data pada index 4.

0	1	2	3	4	5
5	4	7	3	6	9




4. Langkah ke-3, data pada index 4 dan 5 sudah terurut. Dari yang tersisa ditemukan data terbesar yaitu 6 pada posisi index 2 kemudian ditukar dengan data pada index 3.

0	1	2	3	4	5
5	4	6	3	7	9




5. Langkah ke-4, data pada index 3, 4, dan 5 sudah terurut. Dari yang tersisa ditemukan data terbesar yaitu 5 pada posisi index 0 kemudian ditukar dengan data pada index 2.

0	1	2	3	4	5
5	4	3	6	7	9



6. Langkah ke-5, data pada index 2, 3, 4, dan 5 sudah terurut. Dari yang tersisa ditemukan data terbesar yaitu 4 pada posisi index 1 kemudian ditukar dengan data pada index 1, yaitu index-nya sendiri.

0	1	2	3	4	5
3	4	5	6	7	9



7. Semua data sudah terurut

0	1	2	3	4	5
3	4	5	6	7	9

Berikut ini adalah potongan kode untuk menjalankan proses di atas.

```
const int SIZE{6};
...
for(int i = 0; i < SIZE; ++i)
{
    idxs = 0;

    for(int j = 0; j < SIZE - i; ++j)
        if(MyArray[j] > MyArray[idxs])
            idxs = j;

    int tmp = MyArray[idxs];
    MyArray[idxs] = MyArray[size - i];
    MyArray[size - i] = tmp;
}
```

Untuk melakukan pengurutan turun dapat dilakukan dengan cara sebaliknya, pada kondisi:

```
if(MyArray[j] > MyArray[idxs])
```

tanda lebih besar (>) diganti dengan tanda lebih kecil (<)

### 5.2.2 Bubble Sort

Metode bubble mengadopsi dari sifat gelembung air, di mana gelembung yang lebih besar akan lebih mudah dan lebih cepat mencapai permukaan. Prosesnya adalah dengan membandingkan dua demi dua data yang saling berdekatan. Bila data di kiri lebih besar (atau lebih kecil) maka akan ditukar. Proses akan terus berlangsung hingga semua data “digelembungkan” sesuai urutan besarnya nilai.

Dengan contoh array yang sama, MyArray, seperti metode sebelumnya, ilustrasi pengurutan naik menggunakan metode bubble adalah sebagai berikut.

1. Kondisi awal

0	1	2	3	4	5
5	9	7	3	6	4

2. Langkah ke-1, menggelembungkan data terbesar ke-1

0	1	2	3	4	5
5	9	7	3	6	4

0	1	2	3	4	5
5	9	7	3	6	4



0	1	2	3	4	5
5	7	9	3	6	4



0	1	2	3	4	5
5	7	3	9	6	4



0	1	2	3	4	5
5	7	3	6	9	4

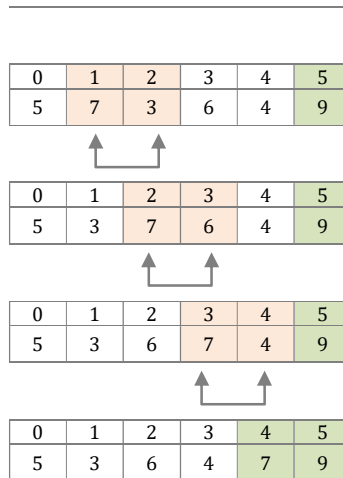


0	1	2	3	4	5
5	7	3	6	4	9

Didapat data terurut 9

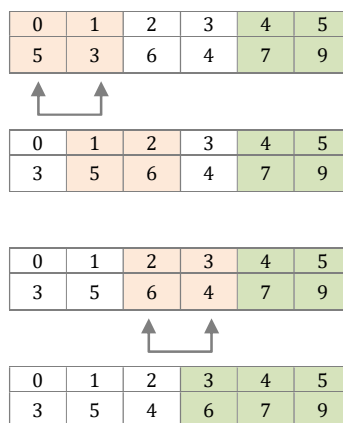
3. Langkah ke-2, menggelembungkan data terbesar ke-2

0	1	2	3	4	5
5	7	3	6	4	9



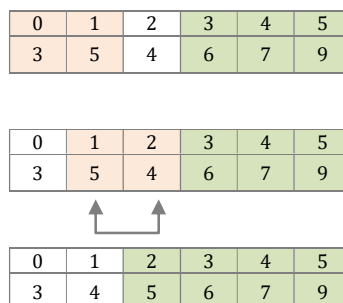
Didapat data terurut 7 dan 9

4. Langkah ke-3, menggelembungkan data terbesar ke-3



Didapat data terurut 6, 7, dan 9

5. Langkah ke-4, menggelembungkan data terbesar ke-4



Didapat data terurut 5, 6, 7, dan 9

6. Langkah ke-5, menggelembungkan data terbesar ke-5

0	1	2	3	4	5
3	4	5	6	7	9

0	1	2	3	4	5
3	4	5	6	7	9

Didapat data terurut 4, 5, 6, 7, dan 9

7. Kondisi akhir semua data sudah terurut

0	1	2	3	4	5
3	4	5	6	7	9

Potongan kode untuk proses di atas adalah sebagai berikut.

```
const int SIZE{6};
...
for(int i = 0; i < SIZE; ++i)
{
    for(int j = 0; j < SIZE - i - 1; ++j)
    {
        if(MyArray[j] > MyArray[j + 1])
        {
            int tmp = MyArray[j];
            MyArray[j] = MyArray[j + 1];
            MyArray[j + 1] = tmp;
        }
    }
}
```

Untuk melakukan pengurutan turun dapat dilakukan dengan cara sebaliknya, pada kondisi:

```
if(MyArray[j] > MyArray[j + i])
```

tanda lebih besar (>) diganti dengan tanda lebih kecil (<)

### 5.2.3 Insertion Sort

Metode insertion mengadopsi dari kehidupan nyata, seperti pengurutan dokumen pada filing cabinet. Prosesnya adalah dengan cara mengambil satu demi satu data yang masih acak untuk dicarikan posisi yang sesuai di deratan data yang sudah terurut kemudian disisipkan. Proses akan terus berlangsung hingga semuanya berada pada posisi yang sudah sesuai.

Masih menggunakan contoh array yang sama, MyArray, seperti sebelumnya, proses pengurutan menggunakan metode insertion dapat digambarkan seperti berikut.

Kondisi awal, menganggap data pada index 0 sudah terurut.

0	1	2	3	4	5
5	9	7	3	6	4

Didapat data terurut 5

Langkah ke-1, mencari posisi untuk data pada index 1, yaitu 9

0	1	2	3	4	5
5	9	7	3	6	4

Data 9 sudah pada posisinya sehingga masih tetap menempati posisi index 1. Didapat data terurut 5 dan 9

Langkah ke-2, mencari posisi untuk data pada index 2, yaitu 7

0	1	2	3	4	5
5	9	7	3	6	4

→

Data 7 diposisikan pada index 1 sehingga terjadi pergeseran data. Didapat data terurut 5, 7, dan 9

Langkah ke-3, mencari posisi untuk data pada index 3, yaitu 3

0	1	2	3	4	5
5	7	9	3	6	4

→ → →

Data 3 diposisikan pada index 0 sehingga terjadi pergeseran data. Didapat data terurut 3, 5, 7, dan 9

Langkah ke-4, mencari posisi untuk data pada index 4, yaitu 6

0	1	2	3	4	5
3	5	7	9	6	4

→ →

Data 6 diposisikan pada index 2 sehingga terjadi pergeseran data. Didapat data terurut 3, 5, 6, 7, dan 9

Langkah ke-5, mencari posisi untuk data pada index 4, yaitu 4

0	1	2	3	4	5
3	5	6	7	9	4

→ → → →

Data 4 diposisikan pada index 1 sehingga terjadi pergeseran data. Didapat data terurut 3, 4, 5, 6, 7, dan 9

Kondisi akhir semua data sudah terurut

0	1	2	3	4	5
3	4	5	6	7	9

Berikut ini adalah potongan kode untuk menjalankan proses di atas.

```

const int SIZE{6};
...
for(int i = 1; i < SIZE; ++i)
{
    int tmp = MyArray[i];
    int itmp = i;

    while(itmp > 0 && tmp < MyArray[itmp - 1])
        MyArray[itmp] = MyArray[--itmp];

    MyArray[itmp] = tmp;
}

```

Untuk melakukan pengurutan turun dapat dilakukan cara sebaliknya, pada kondisi

```
tmp < MyArray[itmp - 1]
```

tanda lebih kecil (<) diganti dengan tanda lebih besar (>)

### 5.3 Standard Library `std::sort()`

Seperti telah diuraikan pada bab *searching* sebelumnya (§5.3), dalam praktek pemrograman menggunakan C++ yang sesungguhnya seorang pemrogram tidak harus mengimplementasikan sendiri algoritma *sorting* untuk keperluan pengurutan data. C++ menyediakan *standard library* berupa fungsi `std::sort()` untuk keperluan pengurutan data dalam *array* atau *sortable data structure* lain yang lebih kompleks. Definisi fungsi ini terdapat dalam header `<algorithm>`.

Pembahasan lebih lanjut mengenai definis dan penggunaan tentang fungsi ini akan diuraikan di bab selanjutnya tentang *Standard Template Library (STL)* (§10.4).

### 5.4 Contoh Kasus

Program untuk membandingkan kecepatan pengurutan metode selection, bubble, dan insertion. Data dalam array di-generate secara acak. Tiap metode akan dicatat waktu sebelum dan sesudah proses pengurutan yang kemudian akan diperoleh selisih waktu dalam satuan milisecond. Berikut ini contoh tampilan program waktu berjalan.

#### Perbandingan Metode Pengurutan

Data Acak:

41 44 43 12 89 49 57 88 76 90 46 11 61 6 45 33 59 97 3 42 79 78 83 45 28

Bubble Sort

data terurut:

3 6 11 12 28 33 41 42 43 44 45 45 46 49 57 59 61 76 78 79 83 88 89 90 97

waktu: 5 milisecond

-----

Selection Sort

data terurut:

3 6 11 12 28 33 41 42 43 44 45 45 46 49 57 59 61 76 78 79 83 88 89 90 97

waktu: 4 milisecond

-----

Insertion Sort



```
data terurut:
3 6 11 12 28 33 41 42 43 44 45 45 46 49 57 59 61 76 78 79 83 88 89 90 97
waktu: 2 milisecond
-----
```

## Penyelesaian

```
/*
 * program perbandingan metode sorting:
 * Bubble, Selection, dan Insertion
 *
 * Bayu Setiaji 2013
 * http://yipsoft.com
 *
 */

#include <iostream>
#include <ctime>
#include <windows.h>

using namespace std;

const int SIZE{25};
const int BUBBLE{0};
const int SELECTION{1};
const int INSERTION{2};

int nsrc[SIZE];
int ndata[SIZE];

void init_data();
void load_data();
void view_data();
void bubsort();
void selsort();
void inssort();
void run_sort(int);

int main(void) {
    cout << "Perbandingan Metode Pengurutan\n";

    init_data();
    load_data();

    cout << "Data Acak:\n";
    view_data();
    cout << "\n\n";

    // Bubble Sort
    cout << "Bubble Sort\n";
    load_data();
    run_sort(BUBBLE);

    // Selection Sort
    cout << "Selection Sort\n";
    load_data();
    run_sort(SELECTION);

    // Insertion Sort
    load_data();
    cout << "Insertion Sort\n";
    run_sort(INSERTION);
}
```

```

    return 0;
}

void init_data(void)
{
    srand(time(NULL));
    for(int i = 0; i < SIZE; ++i)
        nsrc[i] = rand() % 100;
}

void load_data(void)
{
    for(int i = 0; i < SIZE; ++i)
        ndata[i] = nsrc[i];
}

void view_data(void)
{
    for(int i = 0; i < SIZE; ++i)
        printf("%d ", ndata[i]);

    cout << endl;
}

void bubsort(void)
{
    for(int i = 0; i < SIZE - 1; ++i)
    {
        for(int j = 0; j < SIZE - i - 1; ++j)
        {
            if(ndata[j] > ndata[j + 1])
            {
                int tmp = ndata[j];
                ndata[j] = ndata[j + 1];
                ndata[j + 1] = tmp;
                Sleep(10);
            }

            Sleep(10);
        }

        Sleep(10);
    }
}

void selsort(void)
{
    for(int i = 0; i < SIZE; ++i)
    {
        int min = i;
        for(int j = i; j < SIZE; ++j)
        {
            if(ndata[j] < ndata[min])
            {
                min = j;
                Sleep(10);
            }

            Sleep(10);
        }

        int tmp = ndata[i];
        ndata[i] = ndata[min];
        ndata[min] = tmp;
    }
}

```

```

        Sleep(10);
    }
}

void inssort(void)
{
    for(int i = 1; i < SIZE; ++i)
    {
        int m = ndata[i];
        s = i;

        while(s >= 0 && m < ndata[s - 1])
        {
            ndata[s] = ndata[--s];
            Sleep(10);
        }

        ndata[s] = m;
        Sleep(10);
    }
}

void run_sort(int m)
{
    auto t1 = time(NULL);

    switch(m)
    {
        case 0: bubsort(); break;
        case 1: selsort(); break;
        case 2:
            default: inssort(); break;
    }

    auto t2 = time(NULL);
    auto t = t2 - t1;

    cout << "data terurut:\n";
    view_data();
    cout << "waktu: " << t << " milisecond\n";
    cout << "-----\n\n";
}

```

## 5.5 Latihan

1. Di antara 3 metode pengurutan yang sudah dibahas, manakah yang paling cepat? Mengapa?
2. Berdasarkan proses perbandingan dan pertukarannya, manakah yang lebih cepat antara metode bubble dan selection untuk menangani data yang hampir terurut? Mengapa?

Bila memungkinkan, modifikasilah proses pengurutan 3 metode yang sudah dibahas di atas ke dalam bentuk rekursif.