**PRAKTIKUM V**

**SORTING**

**5.1. Pendahuluan**

Kumpulan data biasanya diurutkan dalam urutan tertentu untuk memudahkan proses selanjutnya. Misalnya buku telepon, selalu diurutkan berdasarkan nama pemilik telepon. Contoh lain, daftar hadir mahasiswa pada suatu matakuliah, selalu diurutkan berdasarkan nomor mahasiswa. Keadaan data yang terurut ini memudahkan user untuk menemukan suatu data nantinya.

Pada mulanya, tentu saja data tersebut tidak terurut. Orang yang datang meminta sambungan telepon tidak terurut berdasarkan namanya. Mahasiswa yang mendaftar suatu matakuliah juga tidak diatur berdasarkan nomor mahasiswanya. Dengan demikian diperlukan suatu algoritma yang dapat mengatur data yang tadinya tidak terurut menjadi data yang terurut. Algoritma itu disebut algoritma sorting.

Algoritma sorting berisi teknik untuk memeriksa data dalam array dan mengaturnya dalam urutan tertentu, seperti urut naik (ascending order) atau urut turun (descending order). Hasil dari algoritma sorting adalah data yang sudah terurut dalam array sedemikian sehingga:

i1 <= i2 <= …. <= in untuk urut naik,

atau i1 >= i2 >= …. >= in  untuk urut turun.

Terdapat banyak metode sorting yang sudah dibuat orang, diantaranya adalah exchange sort, selection sort, insertion sort, quick sort, Shell sort, dan merge sort. Algoritma sorting ini mempunyai cara yang berbeda-beda untuk mengurutkan data dalam array. Berdasarkan tingkat kerumitannya, algoritma sorting dibedakan menjadi algoritma yang sederhana dan yang kompleks. Algoritma exchange sort, selection sort, dan insertion sort termasuk dalam algoritma yang sederhana, dan algoritma yang lainnya termasuk dalam algoritma kompleks. Pada umumnya algoritma sorting yang kompleks merupakan pengembangan dari algoritma sorting yang sederhana. Untuk memudahkan pembahasan algoritma sorting ini, diasumsikan data yang digunakan bertipe integer dan disimpan dalam array, lalu akan diurutkan menjadi urut naik.

* 1. **Exchange Sort**

Sesuai namanya, exchange yang berarti menukar atau mengganti, algoritma ini banyak menggunakan teknik pertukaran dalam proses mengurutkan datanya. Exchange sort termasuk dalam algoritma sorting yang sederhana. Secara umum algoritmanya dapat dilihat pada gambar 5.1 di bawah ini

**Void Exchange\_Sort(data[], n):**

1. **For (i = 0 ; i < n ; i++)**

**For (j = n ; j >= i ; j--)**

**If (data[j-1] > data[j])**

**{ temp = data[j-1]**

**data[j-1] = data[j]**

**data[j] = temp**

**}**

## 2. Return

Gambar 5.1 Algoritma Bubble Sort

Implementasinya dalam program C++ dapat dilihat pada gambar 5.2 di halaman berikut ini. Program tersebut digunakan untuk mengurutkan 9 data integer. Eksekusi program dan lihat hasilnya. Output program adalah array yang berisi data yang sudah terurut.

Untuk melihat proses pengurutan data dengan algoritma Exchange sort ini, tambahkan fungsi tampilkan(data[],n) setelah for loop yang di dalam dieksekusi. Perhatikan output dari proses pengurutan secara bertahap. Dapat dilihat bahwa data yang besar bergerak dengan perlahan ke tempat yang sebenarnya. Seperti gelembung udara yang bergerak ke atas, maka algoritma exchange sort ini dikenal juga dengan nama Bubble Sort.

Dari output, juga dapat dilihat bahwa setelah loop ke 6, data dalam array sudah terurut. Tetapi proses pengurutan tidak berhenti disitu, berlanjut terus sampai loop ke 8. Hal ini sesuai dengan statement pada for loop yang di luar: **for (int i=0 ; i<n ; i++)**

Dengan demikian, eksekusi for loop dilakukan sebanyak 9 kali tanpa memperhatikan apakah data dalam array sudah terurut atau belum.

Supaya program dapat mengetahui jika data sudah terurut maka digunakan flag atau tanda. Tanda yang digunakan adalah terjadi pertukaran atau tidak. Jika data belum terurut maka akan terjadi pertukaran, tetapi jika data sudah terurut maka tidak terjadi pertukaran. Dengan demikian program dimodifikasi dengan menggunakan while loop yang akan memeriksa tanda. Jika terjadi pertukaran maka tanda = true, data belum terurut, lakukan proses pengurutan. Jika tidak terjadi pertukaran maka tanda = false, data sudah terurut, proses pengurutan berhenti. Modifikasi algoritma exchange sort dapat dilihat pada gambar 5.3.

**#include <iostream.h>**

**void exch(int[], int); //deklarasi fungsi**

**void tampilkan(int[], int);**

**int main()**

**{**

**int bil[9] = {7, 2, 5, 1, 4, 9, 6, 8, 3};**

**cout<<"Bilangan yang tidak terurut: ";**

**tampilkan(bil, 9);**

**cout<<endl;**

**exch(bil,9);**

**cout<<endl;**

**cout<<"Bilangan yang sudah terurut: ";**

**tampilkan(bil, 9);**

**return 0;**

**}**

**void exch(int data[], int n) // definisi fungsi**

**{ //exchange sort**

**int temp;**

**for (int i=0 ; i<n ; i++)**

**{**

**for (int j=n ; j>i ; j--)**

**if (data[j-1] > data[j])**

**{ temp = data[j-1]; //tukar data**

**data[j-1] = data [j];**

**data[j] = temp;**

**}**

**// cout<<"Array setelah loop ke "<<i<<" adalah ";**

**// tampilkan(data,n);**

**}**

**}**

**void tampilkan(int data[], int n) // definisi fungsi**

**{ // untuk menampilkan isi array**

**for(int i=0 ; i<n ; i++)**

**cout<<data[i]<< " , ";**

**cout<<endl;**

**}**

Gambar 5.2 Contoh Program Exchange Sort

Hasil eksekusi program pada gambar 5.3 menunjukkan bahwa proses pengurutan berhenti setelah data dalam array terurut, bandingkan dengan hasil eksekusi program pada gambar 5.2.

**5.3. Selection Sort** Algoritma exchange sort kurang efisien jika diterapkan pada data yang banyak karena data hanya berpindah 1 elemen setiap kali terjadi pertukaran. Untuk memaksimalkan perpindahan elemen, dapat digunakan algoritma selection sort.

**#include <iostream.h>**

**void exch(int[], int); //deklarasi fungsi**

**void tampilkan(int[], int);**

**int main()**

**{**

**int bil[9] = {7, 2, 5, 1, 4, 9, 6, 8, 3};**

**cout<<"Bilangan yang tidak terurut: ";**

**tampilkan(bil, 9);**

**cout<<endl;**

**exch(bil,9);**

**cout<<endl;**

**cout<<"Bilangan yang sudah terurut: ";**

**tampilkan(bil, 9);**

**return 0;**

**}**

**void exch(int data[], int n) // definisi fungsi**

**{ //exchange sort dengan flag**

**bool tukar;**

**int temp,i;**

**i = 0;**

**do**

**{**

**tukar = false;**

**for (int j=n ; j>0 ; j--)**

**if (data[j-1] > data[j])**

**{ temp = data[j-1];**

**data[j-1] = data [j];**

**data[j] = temp;**

**tukar = true;**

**}**

**cout<<"Array setelah loop ke "<< i <<" adalah ";**

**tampilkan(data,n);**

**i=i+1;**

**} while(tukar);**

**}**

**void tampilkan(int data[], int n) // definisi fungsi**

**{ // untuk menampilkan isi array**

**for(int i=0 ; i<n ; i++)**

**cout<<data[i]<< " , ";**

**cout<<endl;**

**}**

Gambar 5.3 Program Exchange sort dengan flag

Sesuai namanya, selection yang berarti memilih, algoritma ini memilih data yang akan dipindahkan. Data yang dipilih adalah data yang terkecil. Lalu data ini ditempatkan pada posisi yang pertama jika data tersebut tidak berada di posisi pertama. Kemudian akan dipilih data yang terkecil berikutnya untuk ditempatkan pada posisi kedua. Demikian seterusnya sampai data terakhir. Secara umum algoritma ini dapat dilihat pada gambar 5.4 di bawah ini.

**Void Selection\_sort(data[], n)**

**1. For (i = 0; i < n-1 ; i++)**

**{**

**Min = data[i]**

**Pos = i**

**For (j = i+1 ; j < n ; j++) //cari data terkecil**

**{ If (Min > data[j])**

**{ Min = data[j]**

**Pos = j**

**}**

}

**temp = data[i] //tukar data terkecil**

**data[i] = data[j] //dengan data pada indeks i**

**data[j] = temp**

**}**

**2. Return**

Gambar 5.4 Algoritma Selection Sort

Buatlah program C++ untuk mengimplementasikan algoritma di atas. Lalu tambahkan statement untuk menampilkan proses pengurutannya.

Sama seperti Exchange sort, algoritma selection sort terdiri dari 2 struktur loop. Loop yang di luar untuk menentukan posisi data yang sudah terurut, loop yang di dalam untuk mencari data dengan nilai yang terkecil dalam array. Jika ditemukan, nilainya disimpan dalam variabel Min dan indeksnya disimpan dalam variabel Pos. Setelah loop yang di dalam selesai, nilai yang terkecil tadi (variabel Min) beserta indeksnya (variabel Pos) ditukar dengan nilai dan indeks pada variabel pengontrol loop yang di luar (data pada indeks i) lalu indeks i di-increment. Demikian seterusnya sampai ke akhir array.

Berbeda dari exchange sort yang dapat mendeteksi jika data dalam array sudah terurut sehingga proses pengurutan berhenti, pada selection sort proses mencari bilangan terkecil akan terus dilakukan meskipun data sudah terurut (lihat kembali hasil eksekusi program). Dengan demikian selection sort bekerja dalam waktu konstan apapun kondisi datanya.

**5.4. Insertion Sort** Sesuai namanya, insertion yang berarti menyisipkan, algoritma ini akan menggeser elemen array pada indeks i ke i +1 jika data yang baru lebih kecil dari data pada indeks i lalu menyisipkan data yang baru pada tempat kosong hasil pergeseran. Mula-mula ditetapkan, data pada indeks pertama, sudah terurut. Lalu lihat indeks berikutnya, jika lebih besar, maka sudah terurut, jika lebih kecil, geser data, lalu sisipkan data yang baru di tempat yang kosong. Algoritma insertion sort dapat dilihat pada gambar 5.5 di bawah ini.

**Void Insertion\_Sort(data[],n)**

**1. For (i = 1 ; i < n ; i++)**

**{**

**x = data[i]**

**j = i-1**

**while (x < data[j])**

**{**

**data[j+1]= data[j]**

**j = j-1**

**}**

**data[j+1] 🡨 x**

**2. return**

Gambar 5.5 Algoritma Insertion Sort

Sama seperti algoritma yang lain, algoritma insertion sort juga terdiri dari 2 loop. Loop yang di luar mengontrol banyaknya data yang sudah terurut, loop yang di dalam memeriksa apakah data sudah pada posisinya atau belum, mulai dari indeks j. Jika data sudah pada posisinya, maka tidak ada pergeseran. Jika belum, geser data mulai dari posisi j ke j+1 sampai ditemukan posisi data yang benar lalu sisipkan data ditempatnya.

Perbedaan insertion sort dengan kedua algoritma yang lain adalah pada banyaknya data yang harus diperiksa pada loop yang di dalam. Untuk kedua algoritma sebelumnya, seluruh data yang berada dalam range akan diperiksa dan diproses. Tetapi pada insertion sort, tidak seluruh data dalam range yang diproses, hanya data yang tidak pada posisinya saja yang diproses. Sama seperti algoritma selection sort, algoritma insertion sort tidak dapat mendeteksi apakah data dalam array sudah dalam kondisi terurut atau belum, sehingga loop yang di luar harus dieksekusi untuk seluruh elemen dalam array

**Latihan Soal Sorting Sederhana**

1. Analisis algoritma sorting yang sederhana

Untuk mengetahui mana dari ke 3 algoritma yang lebih efisien dalam melakukan proses pengurutan data, dapat dilakukan dengan menghitung banyaknya proses perbandingan dan pertukaran/pemindahan data selama algoritma dieksekusi. Dalam program yang sudah dibuat, tambahkan statement untuk menghitung banyaknya perbandingan dan pertukaran/pergeseran data dari masing-masing algoritma.

1. Untuk mengetahui perilaku algoritma terhadap data yang diproses, buatlah array yang datanya terurut dan array yang datanya terurut kebalikan. Lalu eksekusi kedua array dengan ke 3 algoritma tadi. Hitung pula banyaknya perbandingan dan pertukaran/pergeseran yang terjadi. Algoritma mana yang paling efisien untuk suatu kondisi data? Jelaskan.

**5.5. Quicksort**

Quicksort merupakan pengembangan dari exchange sort, disebut juga Partition-Exchange sort. Algortima ini merupakan salah satu contoh dari metoda divide and conquer, yaitu dengan memilah-milah data yang akan diurutkan sehingga tiap sub proses pengurutan hanya melibatkan sebagian data. Pemilahan data sangat tergantung dari data yang menjadi pivot atau data pemisah. Jika pivot dapat membagi data yang akan diurutkan menjadi 2 bagian yang kira-kira sama besar maka algoritma ini akan menjadi efisien, tetapi jika banyaknya data tidak terbagi dengan seimbang maka algoritma ini menjadi seperti exchange sort. Algoritma ini menggunakan metode rekursif dalam proses pengurutan datanya, contoh algoritma dapat dilihat pada gambar 5.7 di bawah ini.

void Quicksort (int data[],int left,int right)

{

int mid, tmp;

i = left;

j = right;

mid = data[(left + right)/2];

do {

while(data[i] < mid)

i++;

while(mid < data[j])

j--;

if (i <= j)

{

tmp = data[i];

data[i] = data[j];

data[j] = tmp;

i++;

j--;

}

} while (i <= j);

if (left < j) Quicksort(data,left,j);

if (i < right) Quicksort(data,i,right);

}

Gambar 5.6 Algoritma Quick Sort

**5.6. Merge Sort**

Sesuai namanya, algoritma merge sort menggabungkan (merge) 2 deretan data (A[n] dan B[m]) yang masing-masing sudah terurut menjadi 1 deretan data (C[m+n]) yang terurut. Algoritma ini cukup sederhana karena hanya membandingkan 2 data. Data yang lebih kecil akan berada di depan, dan data yang lebih besar akan berada di belakang. Algortima merge sort dapat dilihat pada gambar 5.7.

void SIMPLE\_MERGE(A, B, C, N, M)

{

I = 1;

J = 1;

K = 1;

while ((I <= N) && (J <= M))

{

if ( A[I] <= B[J] )

{ C[K] = A[I];

K = K + 1;

I = I + 1;

}

else

{ C[K] = B[J];

K = K + 1;

J = J + 1;

}

}

while ( I <= N )

{ C[K] = A[I];

K = K + 1;

I = I + 1;

}

while ( J <= M )

{ C[K] = B[J];

K = K + 1;

J = J + 1;

}

}

Gambar 5.7 Algoritma Merge Sort

Untuk mengurutkan data dalam sebuah array, algoritma simple merge dimodifikasi dengan membagi array menjadi subarray- subarray lalu menggabungkan subarray tersebut. Prosesnya dilakukan secara rekursif dengan membagi array menjadi 2 subarray, lalu tiap subarray dibagi lagi sampai banyaknya elemen tiap subarray = 1 lalu digabungkan dengan metode mergesort. Algoritma modifikasi simple merge dapat dilihat pada gambar 5.8 di bawah ini.

#include <iostream.h>

#include <conio.h>

void mers(int[], int, int); //deklarasi fungsi

void tampilkan(int[], int);

int main()

{

int bil[16] = {27, 12, 25, 32, 21, 14, 29, 16, 18, 13, 17, 22, 15, 11, 24, 19, };

cout<<"Array mula-mula: ";

tampilkan(bil, 16);

cout<<endl;

mers(bil, 0, 15);

cout<<"Array akhir: ";

tampilkan(bil, 16);

return 0;

}

void tampilkan(int data[], int n) // definisi fungsi

{ // untuk menampilkan isi array

for (int i=0 ; i<n ; i++)

cout<<data[i]<< ", ";

cout<<endl;

}

void mers(int a[], int l, int r)

{

int i, j, k, m;

int b[16];

if (r > l)

{

m = (r + l)/2;

mers(a, l, m);

mers(a, m+1, r);

for (i=m+1 ; i > l ; i--)

b[i-1] = a[i-1];

for (j=m ; j < r ; j++)

b[r+m-j] = a[j+1];

for (k=l ; k<=r ; k++)

{

if (b[i] < b[j])

{ a[k] = b[i];

i++;

}

else

{ a[k] = b[j];

j--;

}

}

}

}

**Latihan soal**

1. Buatlah program utama untuk quicksort dan merge sort di atas lalu jalankan progam anda.
2. Tambahkan dalam program saudara fungsi untuk membangkitkan bilangan acak integer. Buatlah array untuk menyimpan 10000, 20000, 40000 dan 80000 data.Catat waktu yang dibutuhkan untuk mengurutkan data di atas.

**#include <iostream.h>**

**//void sele(int[], int); //deklarasi fungsi**

**void inse(int[], int);**

**void tampilkan(int[], int);**

**int main()**

**{**

**int bil[9] = {7, 2, 5, 1, 4, 9, 6, 8, 3};**

**cout<<"Bilangan yang tidak terurut: ";**

**tampilkan(bil, 9);**

**cout<<endl;**

**// sele(bil,9);**

**inse(bil,9);**

**cout<<endl;**

**cout<<"Bilangan yang sudah terurut: ";**

**tampilkan(bil, 9);**

**return 0;**

**}**

**/\***

**void sele(int data[], int n) // definisi fungsi**

**{ //selection sort**

**int Min,Pos,temp;**

**for (int i = 0 ; i < n-1 ; i++)**

**{**

**Min = data[i];**

**Pos = i;**

**for (int j = i+1 ; j < n ; j++) //cari data**

**{ if (Min > data[j]) //terkecil**

**{ Min = data[j];**

**Pos = j;**

**}**

**}**

**temp = data[i]; //tukar data**

**data[i] = data[Pos];**

**data[Pos] = temp;**

**cout<<"Array setelah loop ke "<< i <<" adalah ";**

**tampilkan(data,n);**

**}**

**}**

**\*/**

**void inse(int data[], int n) // definisi fungsi**

**{ //insertion sort**

**int x, j;**

**for (int i = 1 ; i < n ; i++)**

**{**

**x = data[i];**

**j = i-1;**

**while (x < data[j])**

**{**

**data[j+1] = data[j];**

**j = j-1;**

**}**

**data[j+1] = x;**

**cout<<"Array setelah loop ke "<< i <<" adalah ";**

**tampilkan(data,n);**

**}**

**}**

**void tampilkan(int data[], int n) // definisi fungsi**

**{ // untuk menampilkan isi array**

**for(int i=0 ; i<n ; i++)**

**cout<<data[i]<< " , ";**

**cout<<endl;**

**}**