Nama : Fisma Meividianugraha Subani

NIM : 21091397017

Kelas : 2021 A

LAPORAN INDIVIDU INSERTION SORT

1. Program C++ Insertion Sort

A. Fungsi Void Insertion Sort (Bagian Pertama)

```
// Implementasi Program C++ Insertion Sort
#include <iostream>
using namespace std;


// Fungsi Void Insertion_Sort untuk mengurutkan array
void Insertion_Sort(int array[], int nilai)
{
    /* Deklarasi variabel loop dan j digunakan dalam perulangan.
    Variabel key digunakan untuk penyisipan nilai*/
    int loop, key, j;
    // Looping dari 1 sampai array kurang dari nilai
    for (loop = 1; loop < nilai; loop++)
    {
       // Proses penyisipan nilai (Insert nilai)
        key = array[loop];
        j = loop - 1;
       // Membandingkan nilai lebih kecil dipindah ke depan/kiri (Sort nilai dari
       kecil ke besar)
        while (j >= 0 && array[j] > key)
        {
            array[j + 1] = array[j];
            j = j - 1;
        }
        array[j + 1] = key;
    }
}
```

B. Fungsi Void Cetak Array (Bagian Kedua)

```
// Fungsi Void Cetak_Array untuk menampilkan hasil sorting array
void Cetak_Array(int array[], int nilai)
{
    // Menampilkan kalimat
    cout << "Hasil Array yang sudah di Sorting adalah : "<<endl;
    // Deklarasi variabel loop digunakan untuk perulangan
    int loop;
    // Looping dari 0 sampai array kurang dari nilai
    for (loop = 0; loop < nilai; loop++)
    {
       // Menampilkan hasil sorting array [] (terurut dari kecil ke besar)
       cout <<"["<< array[loop]<<"]";
    }
    cout << endl;
}
```

C. Fungsi Main Output Program (Bagian Ketiga)

```
// Fungsi main untuk mencetak output program
int main()
{
	// Deklarasi variabel nilai sebagai input masukan
	int nilai;
	// Input batas jumlah array (Input pertama)
	cout << "Masukan Batas Jumlah Array :";
	cin >> nilai;
	cout<<endl;

	// Input nilai array berdasarkan looping (Input kedua)
	cout << "Masukan Nilai Array Indeks " <<loop<<" : ";
	int array[nilai];

	// Looping dari 0 sampai array kurang dari nilai
	for(int loop=0; loop < nilai; loop++)
	{
		cin >> array[loop];
	}
	cout<<endl;

	// Memanggil fungsi void Insertion_Sort dengan variabel array dan nilai
	Insertion_Sort(array, nilai);
	// Memanggil fungsi void Cetak_Array dengan variabel array dan nilai
	Cetak_Array(array, nilai);
	return 0;
}
```

D. Hasil Output Program

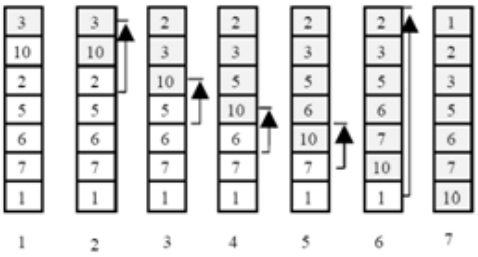
```
Masukan Batas Jumlah Array :5

Masukan 5 Nilai Array :
4 20 3 9 13

Hasil Array yang sudah di Sorting adalah :
[3][4][9][13][20]

--------------------------------
Process exited after 10.96 seconds with return value 0
Press any key to continue . . .
```

Gambar 1. 1 Hasil Output Program

2. Insertion Sort

A. Konsep Dasar Insertion Sort

Insertion sort merupakan sebuah algoritma sederhana yang digunakan sebagai pengurutan sebuah list nilai yang hampir terurut. Konsep sederhana proses pada algoritma insertion sort, yaitu dengan menyisipkan elemen key di posisi yang benar (di antara nilai yang lebih kecil atau sama dengan nilai tersebut). Pada algoritma ini dapat dilakukan penghematan memori, yaitu penerapannya memanfaatkan proses pengurutan di tempat dengan membandingkan elemen saat ini denga elemen sebelumnya yang sudah diurut, kemudian elemen tersebut ditukar sampai pada posisi yang benar. Proses pengurutan dilakukan secara berulang hingga tidak ada elemen yang tersisa pada input. Implementasi Insertion Sort dalam kehidupan sehari-hari yaitu mengurutkan kartu remi di tangan. Pemain berasumsi bahwa kartu pertama sudah diurutkan. Lalu, pemain memilih kartu yang belum diurutkan (sortir). Jika kartu yang belum diurutkan lebih besar dari kartu yang ada di tangan, maka kartu tersebut ditempatkan di sebelah kanan. Langkah berikutnya tetap sama, kartu lain yang belum diurutkan diambil dan diletakkan di tempat yang benar. Berikut adalah contoh gambaran proses dari insertion sort.

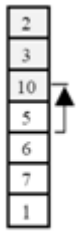


Gambar 2. 1 Proses Insertion Sort

Pembahasan proses insertion sort pada gambar 2.1:

a) Pada pass 1, disajikan sebuah elemen array dengan nilai [3, 10, 2, 5, 6, 7, 1] yang masih belum diurutkan (Sortir) dan elemen pertama (3) diasumsikan terurut. elemen kedua (10) disimpan ke dalam key kemudian dibandingkan dengan elemen pertama (3), tetapi nilai elemen kedua lebih besar (10>3) sehingga masih menempati posisi tetap.

b) Pada pass 2, elemen ketiga (2) disimpan ke dalam key kemudian dibandingkan dengan elemen kedua (10) dan kesatu (3), karena nilai elemen ketiga lebih kecil dari pada nilai elemen kedua dan kesatu (2<3<10) maka posisi elemen ketiga (2) dipindah ke elemen pertama sehingga urutannya menjadi [2, 3, 10, 5, 6, 7, 1] (Pass 3).



3

c) Pada pass 3, elemen keempat (5) disimpan ke dalam key kemudian dibandingkan dengan elemen ketiga (10), karena nilai elemen keempat lebih kecil dari pada nilai elemen ketiga (5<10) maka posisi elemen keempat (5) dipindah ke elemen ketiga. Kemudian elemen ketiga (5) juga dibandingkan dengan elemen kedua dan pertama, tetapi nilai elemen ketiga lebih besar (5>3>2) sehingga proses sorting berhenti dan tetap menempati posisi elemen ketiga (5) dan urutannya menjadi [2, 3, 5, 10, 6, 7, 1] (Pass 4).

4

d) Pada pass 4, elemen kelima (6) disimpan ke dalam key kemudian dibandingkan dengan elemen keempat (10), karena nilai elemen kelima lebih kecil dari pada nilai elemen keempat (6<10) maka posisi elemen kelima (6) dipindah ke elemen keempat. Kemudian elemen keempat (6) juga dibandingkan dengan elemen ketiga, kedua, dan pertama, tetapi nilai elemen keempat lebih besar (6>5>3>2) sehingga proses sorting berhenti dan tetap menempati posisi elemen keempat (6) dan urutannya menjadi [2, 3, 5, 6, 10, 7, 1] (Pass 5).

5

e) Pada pass 5, elemen keenam (7) disimpan ke dalam key kemudian dibandingkan dengan elemen kelima (10), karena nilai elemen keenam lebih kecil dari pada nilai elemen kelima (7<10) maka posisi elemen keenam (7) dipindah ke elemen kelima. Kemudian elemen kelima (7) juga dibandingkan dengan elemen keempat, ketiga, kedua, dan pertama, tetapi nilai elemen kelima lebih besar (7>6>5>3>2) sehingga

proses sorting berhenti dan tetap menempati posisi elemen kelima (7) dan urutannya menjadi [2, 3, 5, 6, 7, 10, 1] (Pass 6).

6

f) Pada pass 6, elemen ketujuh (1) disimpan ke dalam key kemudian dibandingkan dengan elemen keenam (10), karena nilai elemen ketujuh lebih kecil dari pada nilai elemen keenam (1<10) maka posisi elemen keenam (7) dipindah ke elemen keenam. Kemudian elemen keenam (1) juga dibandingkan dengan elemen kelima, keempat, ketiga, kedua, dan pertama, karena nilai elemen keenam lebih kecil dari pada nilai elemen kelima, keempat, ketiga, kedua, dan pertama (1<2<3<5<6<7<10) maka posisi elemen ketujuh (1) dipindah ke elemen pertama sehingga urutannya menjadi [1, 2, 3, 5, 6, 7, 10] (Pass 7).

7

g) Pada pass 7, disajikan hasil elemen array yang sudah diurutkan (sortir) dengan nilai [1, 2, 3, 5, 6, 7, 10].

Secara umum proses insertion sort, pengurutan pada setiap nilainya dilakukan melalui beberapa tahapan: Pertama, nilai awal dimasukkan sembarang, kemudian nilai berikutnya dimasukkan di bagian paling akhir. Kedua, nilai tersebut dibandingkan dengan nilai ke (n-1). Jika belum terurut posisi nilai yang sebelumnya digeser sekali ke kanan terus sampai elemen yang sedang diproses menemukan posisi yang tepat. Ketiga, setiap pergeseran akan mengganti posisi pada nilai berikutnya, namun hal tersebut tidak menjadi persoalan sebab elemen berikutnya sudah diproses lebih dahulu. Pada pass 1 merupakan list nilai yang belum terurut, pass 2 sampai 6 adalah proses menyisipkan elemen nilai (insertion sort), pass 7 yaitu hasil dari list nilai yang sudah diurutkan (sortir). Nilai yang diberi warna abu-abu berarti list nilai tersebut sudah terurut. Sedangkan simbol panah menunjukkan proses penyisipan dan perubahan posisi nilai.

B. Kompleksitas Insertion Sort

Pada insertion sort proses perulangan luar (outer loop) dilakukan sebanyak n-1 kali. Akan tetapi berapa kali perulangan dalam (inner loop) dilakukan tergantung pada input yang dimasukan. Kompleksitas algoritma insertion sort dibagi menjadi dua jenis:

1) Kondisi Best Case (Terbaik)

Pada kasus terbaik (Best Case), tercapai ketika list nilai (elemen-elemen) sudah terurut. Contoh best-case dapat dilihat pada pengurutan list nilai [1, 2, 3, 4] di bawah ini.

1	2	3	4

Tabel 2. 1 Kondisi List Nilai Terurut

Input Array = [1, 2, 3, 4]

#Proses 1

[**1**, **2**, 3, 4] akan menjadi [**1**, **2**, 3, 4]

#Proses 2

[1, **2**, **3**, 4] akan menjadi [1, **2**, **3**, 4]

#Proses 3

[1, 2, **3**, **4**] akan menjadi [1, 2, **3**, **4**]

Proses perbandingan dilakukan hanya untuk memverifikasi urutan list nilai sehingga tidak ada perpindahan elemen (nilai) yang terjadi. Jadi perulangan dalam (inner loop) best case berjalan dengan kompleksitas O(N).

2) Kondisi Worst Case (Terburuk)

Pada kasus terburuk (Worst Case), tercapai ketika list nilai (elemen-elemen) dalam urutan terbalik. Contoh worst-case dapat dilihat pada pengurutan ist nilai [4, 3, 2, 1] di bawah ini.

4	3	2	1

Tabel 2. 2 Kondisi List Nilai Urutan Terbalik

Input Array = [4, 3, 2, 1]

#Proses 1

[**4**, **3**, 2, 1] akan menjadi [**3**, **4**, 2, 1]

#Proses 2

[3, **4**, **2**, 1] akan menjadi [3, **2**, **4**, 1]

[**3**, **2**, 4, 1] akan menjadi [**2**, **3**, 4, 1]

#Proses 3

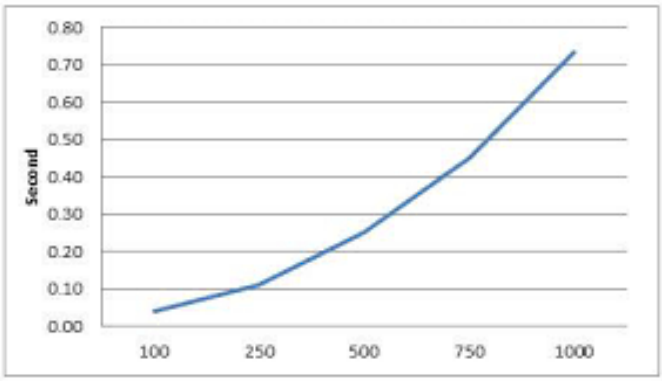
[2, 3, **4**, **1**] akan menjadi [2, 3, **1**, **4**]

[2, **3**, **1**, 4] akan menjadi [2, **1**, **3**, 4]

[**2**, **1**, 3, 4] akan menjadi [**1**, **2**, 3, 4]

Untuk setiap n, elemen nilai [n] selalu dilakukan perbandingan nilai yang lebih kecil dari elemen nilai [0] sampai nilai [n-1], kemudian masing-masing dari elemen dipindahkan satu posisi di depannya sampai menempati posisi yang benar. Jadi perulangan dalam (inner loop) worst case berjalan dengan kompleksitas $O(N^2)$.

3) Grafik Kompleksitas Insertion Sort.



Gambar 2. 2 Grafik Kompleksitas Insertion Sort

C. Kelebihan dan Kekurangan Insertion Sort

Proses insertion sort, mempunyai beberapa kelebihan:

1) Penerapan algoritma yang sederhana.
2) Paling efisien untuk proses sort data berukuran kecil.
3) Algoritma online, yang berarti bisa langsung melakukan sort setiap ada data baru.
4) Proses sort di tempat sehingga tidak membutuhkan memori tambahan untuk menjalankannya.
5) Algoritma stabil dengan proses sorting tercepat pada jumlah elemen yang sedikit.
6) Jika list sudah terurut atau sebagian terurut maka Insertion Sort akan lebih cepat dibandingkan dengan Quicksort.
7) Algoritma insertion sort lebih rapi dibanding Bubble Sort dan Selection Sort.

Proses insertion sort, mempunyai beberapa kekurangan:

1) Tidak disarankan digunakan untuk menangani struktur data dengan lebih dari 2000 elemen.
2) Lebih cocok mensorting bilangan yang bentuknya bulat.
3) Banyaknya operasi yang diperlukan dalam mencari posisi yang tepat untuk elemen larik.
4) Untuk larik yang jumlahnya besar ini tidak praktis.
5) Jika list terurut terbalik sehingga setiap eksekusi dari perintah harus memindai dan mengganti seluruh bagian sebelum menyisipkan elemen berikutnya.
6) Membutuhkan waktu $O(N^2)$ pada data yang tidak terurut, sehingga tidak cocok dalam pengurutan elemen dalam jumlah besar.