

**LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2**

**MODUL XII & XIII
PENGURUTAN DATA**



Disusun Oleh :

Wafiq Nur Azizah / 2311102270

S1IF-11-05

Dosen Pengampu :

Arif Amrulloh, S.Kom., M.Kom.

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. DASAR TEORI

➤ Selection Sort

Selection Sort adalah salah satu algoritma pengurutan sederhana yang bekerja dengan cara mencari elemen terkecil dari sebuah array atau slice, kemudian menempatkannya di posisi awal. Proses ini diulangi untuk elemen-elemen berikutnya hingga seluruh data terurut. Pada setiap iterasi, algoritma ini membandingkan elemen satu per satu untuk menemukan nilai terkecil. Setelah nilai tersebut ditemukan, algoritma akan menukarnya dengan elemen pada indeks awal yang belum terurut. Meski algoritma ini cukup mudah dipahami, efisiensinya rendah untuk dataset besar karena memiliki kompleksitas waktu sebesar $O(n^2)$, dimana n adalah jumlah elemen dalam array.

Dalam bahasa pemrograman Go, selection sort dapat diimplementasikan dengan memanfaatkan struktur sederhana seperti loop for. Meskipun Go dikenal dengan performanya yang baik untuk program berskala besar, algoritma ini lebih cocok digunakan untuk dataset kecil atau sebagai pengenalan pada konsep sorting.

➤ Insertion Sort

Insertion Sort adalah algoritma pengurutan lainnya yang bekerja dengan cara membangun array yang terurut satu per satu. Pada setiap iterasi, elemen dari array yang belum terurut akan diambil dan disisipkan ke posisi yang sesuai dalam bagian array yang sudah terurut. Algoritma ini bekerja mirip dengan cara manusia mengurutkan kartu dalam permainan kartu. Insertion Sort memiliki keunggulan pada dataset kecil atau yang sudah hampir terurut, karena kompleksitas terbaiknya adalah $O(n)$ ketika data hampir sepenuhnya terurut. Namun, kompleksitas terburuknya tetap $O(n^2)$, sehingga kurang efisien untuk dataset besar.

Di Go, Insertion Sort dapat dimanfaatkan untuk aplikasi sederhana di mana efisiensi bukanlah prioritas utama. Algoritma ini lebih intuitif dibandingkan Selection Sort karena melibatkan proses penyisipan yang menyerupai cara berpikir manusia dalam pengurutan manual. Selain itu, algoritma ini juga stabil, artinya tidak mengubah urutan elemen yang bernilai sama dalam array.

II. GUIDED

1. Soal Studi Case (Selection Sort)

Hercules, preman terkenal seantero ibukota, memiliki kerabat di banyak daerah. Tentunya Hercules sangat suka mengunjungi semua kerabatnya itu.

Diberikan masukan nomor rumah dari semua kerabatnya di suatu daerah, buatlah program **rumahkerabat** yang akan menyusun nomor-nomor rumah kerabatnya secara terurut membesar menggunakan algoritma selection sort.

Masukan dimulai dengan sebuah integer **n** ($0 < n < 1000$), banyaknya daerah kerabat Hercules tinggal. Isi **n** baris berikutnya selalu dimulai dengan sebuah integer **m** ($0 < m < 1000000$) yang menyatakan banyaknya rumah kerabat di daerah tersebut, diikuti dengan rangkaian bilangan bulat positif, nomor rumah para kerabat.

Keluaran terdiri dari **n** baris, yaitu rangkaian rumah kerabatnya terurut membesar di masing-masing daerah.

No	Masukan	Keluaran
1	3 5 2 1 7 9 13 6 189 15 27 39 75 133 3 4 9 1	1 2 7 9 13 15 27 39 75 133 189 1 4 9

Keterangan: Terdapat 3 daerah dalam contoh input dan di masing-masing daerah mempunyai 5, 6, dan 3 kerabat.

Sourcecode

```
package main
import "fmt"

//Fungsi untuk mengurutkan array menggunakan Selection Sort
func selectionSort(arr []int, n int) {
    for i := 0; i < n-1; i++ {
        idxMin := i
        for j := i + 1; j < n; j++ {
            //Cari elemen terkecil
            if arr[j] < arr[idxMin] {
                idxMin = j
            }
        }
        //Tukar elemen terkecil dengan elemen di posisi i
        arr[i], arr[idxMin] = arr[idxMin], arr[i]
    }
}
```

```

func main() {
    var n int
    fmt.Print("Masukkan jumlah daerah kerabat (n): ")
    fmt.Scan(&n)

    //Proses tiap daerah
    for daerah := 1; daerah <= n; daerah++ {
        var m int
        fmt.Printf("\nMasukkan jumlah nomor rumah
kerabat untuk daerah %d: ", daerah)
        fmt.Scan(&m)

        //Membaca nomor rumah untuk daerah ini
        arr := make([]int, m)
        fmt.Printf("Masukkan %d nomor rumah kerabat: ",
m)

        for i := 0; i < m; i++ {
            fmt.Scan(&arr[i])
        }

        //Urutkan array dari terkecil ke terbesar
        selectionSort(arr, m)

        //Tampilkan hasil
        fmt.Printf("Nomor rumah terurut untuk daerah %d:
", daerah)
        for _, num := range arr {
            fmt.Printf("%d ", num)
        }
        fmt.Println()
    }
}

```

Screenshoot Output

```

PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Guided\tempCodeRunnerFile.go"
Masukkan jumlah daerah kerabat (n): 3

Masukkan jumlah nomor rumah kerabat untuk daerah 1: 3
Masukkan 3 nomor rumah kerabat: 2 7 9
Nomor rumah terurut untuk daerah 1: 2 7 9

Masukkan jumlah nomor rumah kerabat untuk daerah 2: 3
Masukkan 3 nomor rumah kerabat: 5 2 1
Nomor rumah terurut untuk daerah 2: 1 2 5

Masukkan jumlah nomor rumah kerabat untuk daerah 3: 4
Masukkan 4 nomor rumah kerabat: 4 5 7 1
Nomor rumah terurut untuk daerah 3: 1 4 5 7
PS C:\Users\HP> █

```

Deskripsi Program

Program go di atas menggunakan fungsi `selectionSort` untuk mengurutkan nomor rumah kerabat di berbagai daerah secara menaik (ascending). Pengguna diminta memasukkan jumlah daerah. Kemudian, untuk setiap daerah, diminta memasukkan jumlah nomor rumah dan daftar nomor rumahnya. Setelah itu, nomor rumah diurutkan menggunakan algoritma selection sort. Algoritma ini bekerja dengan mencari angka terkecil dari daftar yang belum terurut, lalu menempatkannya di posisi yang sesuai. Hasil akhirnya adalah daftar nomor rumah yang sudah terurut untuk setiap daerah.

Fungsi `selectionSort` adalah bagian utama program yang bertugas melakukan pengurutan. Fungsi ini memakai dua loop `for`, dimana loop pertama menentukan posisi elemen yang akan diatur sedangkan loop kedua mencari angka terkecil di bagian daftar yang belum diurutkan. Dengan cara ini, angka terkecil secara bertahap dipindahkan ke tempatnya yang benar. Fungsi ini memastikan daftar nomor rumah di setiap daerah diurutkan dari yang terkecil ke yang terbesar sesuai dengan tujuan program.

2. Soal Studi Case (Insertion Sort)

Buatlah sebuah program yang digunakan untuk membaca data integer seperti contoh yang diberikan di bawah ini, kemudian diurutkan (menggunakan metode *insertion sort*) dan memeriksa apakah data yang terurut berjarak sama terhadap data sebelumnya,

Masukan terdiri dari sekumpulan bilangan bulat yang diakhir oleh bilangan negatif. Hanya bilangan non negatif saja yang disimpan ke dalam array.

Keluaran terdiri dari dua baris. Baris pertama adalah isi dari array setelah dilakukan pengurutan, sedangkan baris kedua adalah status jarak setiap bilangan yang ada di dalam array. "Data berjarak x" atau "Data berjarak tidak tetap".

Contoh masukan dan keluaran

No	Masukan	Keluaran
1	31 13 25 43 1 7 19 37 -5	1 7 13 19 25 31 37 43 Data berjarak 6
2	4 40 14 8 26 1 38 2 32 -31	1 2 4 8 14 26 32 38 40 Data berjarak tidak tetap

Sourcecode

```
package main
```

```

import (
    "fmt"
)

// Fungsi untuk mengurutkan array menggunakan Insertion Sort
func insertionSort(arr []int, n int) {
    for i := 1; i < n; i++ {
        key := arr[i]
        j := i - 1

        // Geser elemen yang lebih besar dari key ke
        kanan
        for j >= 0 && arr[j] > key {
            arr[j+1] = arr[j]
            j--
        }
        arr[j+1] = key
    }
}

// Fungsi untuk memeriksa apakah selisih elemen array
tetap
func isConstantDifference(arr []int, n int) (bool, int)
{
    if n < 2 {
        return true, 0
    }

    difference := arr[1] - arr[0]
    for i := 1; i < n-1; i++ {
        if arr[i+1]-arr[i] != difference {
            return false, 0
        }
    }
    return true, difference
}

func main() {
    var arr []int
    var num int

    // Input data hingga bilangan negatif ditemukan
    fmt.Println("Masukkan data integer (akhiri dengan
    bilangan negatif):")
    for {
        fmt.Scan(&num)
        if num < 0 {
            break
        }
        arr = append(arr, num)
    }
}

```

```

n := len(arr)

// Urutkan array menggunakan Insertion Sort
insertionSort(arr, n)

// Periksa apakah selisih elemen tetap
isConstant, difference := isConstantDifference(arr,
n)

// Tampilkan hasil pengurutan
fmt.Println("Array setelah diurutkan:")
for _, val := range arr {
    fmt.Printf("%d ", val)
}
fmt.Println()

// Tampilkan status jarak
if isConstant {
    fmt.Printf("Data berjarak %d\n", difference)
} else {
    fmt.Println("Data berjarak tidak tetap")
}
}

```

Screenshoot Output

```

PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Guided\guided2.go"
14 6 8 4 10 12 16 -2
Array setelah diurutkan:
4 6 8 10 12 14 16
Data berjarak 2
PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Guided\guided2.go"
Masukkan data integer (akhiri dengan bilangan negatif):
31 13 25 14 9 4 -3
Array setelah diurutkan:
4 9 13 14 25 31
Data berjarak tidak tetap
PS C:\Users\HP>

```

Deskripsi Program

Program go di atas bertujuan untuk membaca beberapa bilangan integer dari pengguna, mengurutkannya menggunakan metode insertion sort, dan memeriksa apakah selisih antara elemen-elemen array tersebut tetap sama. Insertion sort adalah teknik pengurutan di mana elemen-elemen dalam array ditempatkan di posisi yang tepat satu per satu. Program membandingkan elemen yang sedang diperiksa dengan elemen sebelumnya dan memindahkan elemen yang lebih besar ke kanan sehingga elemen

tersebut berada di posisi yang benar. Setelah selesai, array akan tersusun dari nilai terkecil ke terbesar yang memudahkan pengguna.

Fungsi utama yang digunakan adalah `insertionSort` yang bertugas mengurutkan array. Fungsi ini menggunakan dua loop yaitu, loop pertama memproses setiap elemen mulai dari indeks 1 dan loop kedua menggeser elemen-elemen yang lebih besar hingga elemen yang sedang diperiksa berada di tempat yang sesuai. Selain itu, ada fungsi `isConstantDifference` yang memeriksa apakah selisih antara elemen-elemen array yang sudah diurutkan selalu sama. Fungsi ini menghitung perbedaan antara elemen-elemen berurutan dan mengecek pola tersebut di seluruh array. Dengan kedua fungsi ini, program dapat mengurutkan bilangan sekaligus memeriksa apakah ada pola selisih yang konsisten.

III. UNGUIDED

1. Soal Studi Case (Selection Sort)

Belakangan diketahui ternyata Hercules itu tidak berani menyeberang jalan, maka selalu diusahakan agar hanya menyeberang jalan sesedikit mungkin, hanya di ujung jalan. Karena nomor rumah sisi kiri jalan selalu ganjil dan sisi kanan jalan selalu genap, maka buatlah program **kerabat dekat** yang akan menampilkan nomor rumah mulai dari nomor yang ganjil lebih dulu terurut membesar dan kemudian menampilkan nomor rumah dengan nomor genap terurut mengecil.

Format **Masukan** masih persis sama seperti sebelumnya.

Keluaran terdiri dari n baris, yaitu rangkaian rumah kerabatnya terurut membesar untuk nomor ganjil, diikuti dengan terurut mengecil untuk nomor genap, di masing-masing daerah.

No	Masukan	Keluaran
1	3 5 2 1 7 9 13 6 189 15 27 39 75 133 3 4 9 1	1 13 12 8 2 15 27 39 75 133 189 8 4 2

Keterangan: Terdapat 3 daerah dalam contoh masukan. Baris kedua berisi campuran bilangan ganjil dan genap. Baris berikutnya hanya berisi bilangan ganjil dan baris terakhir hanya berisi bilangan genap.

Petunjuk:

- Waktu pembacaan data, bilangan ganjil dan genap dipisahkan ke dalam dua array yang berbeda untuk kemudian masing-masing diurutkan tersendiri.
- Atau, tetap disimpan dalam satu array, diurutkan secara keseluruhan. Tetapi pada waktu percentakan, mulai dengan mencetak semua nilai ganjil lebih dulu, kemudian setelah selesai cetaklah semua nilai genapnya.

Sourcecode

```
package main

import "fmt"

func urutkanMeningkat(arr []int, n int) {
    for i := 0; i < n-1; i++ {
        temp := i
        for j := i + 1; j < n; j++ {
            if arr[j] < arr[temp] {
                temp = j
            }
        }
        arr[i], arr[temp] = arr[temp], arr[i]
    }
}
```

```

    }
    }
    arr[i], arr[temp] = arr[temp], arr[i]
}

func urutkanMenurun(arr []int, n int) {
    for i := 0; i < n-1; i++ {
        temp := i
        for j := i + 1; j < n; j++ {
            if arr[j] > arr[temp] {
                temp = j
            }
        }
        arr[i], arr[temp] = arr[temp], arr[i]
    }
}

func main() {
    var jumlahDaerah int
    fmt.Print("Masukkan jumlah daerah kerabat (n): ")
    fmt.Scan(&jumlahDaerah)

    for daerah := 1; daerah <= jumlahDaerah; daerah++ {
        var jumlahRumah int
        var rumah []int

        fmt.Printf("Masukkan jumlah nomor rumah kerabat  
untuk daerah %d: ", daerah)
        fmt.Scan(&jumlahRumah)

        rumah = make([]int, jumlahRumah)
        fmt.Printf("Masukkan %d nomor rumah kerabat: ",  
jumlahRumah)
        for i := 0; i < jumlahRumah; i++ {
            fmt.Scan(&rumah[i])
        }

        rumahGanjil := []int{}
        rumahGenap := []int{}
        for _, nomorRumah := range rumah {
            if nomorRumah%2 != 0 {
                rumahGanjil = append(rumahGanjil,  
nomorRumah)
            } else {
                rumahGenap = append(rumahGenap,  
nomorRumah)
            }
        }

        urutkanMeningkat(rumahGanjil, len(rumahGanjil))
    }
}

```

```

        urutkanMenurun(rumahGenap, len(rumahGenap))

        rumah = append(rumahGanjil, rumahGenap...)

        fmt.Printf("Nomor rumah terurut untuk daerah %d:
", daerah)
        for _, nomor := range rumah {
            fmt.Printf("%d ", nomor)
        }
        fmt.Println()
    }
}

```

Screenshoot Output

```

PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Unguided\unguided1.go"
Masukkan jumlah daerah kerabat (n): 2
Masukkan jumlah nomor rumah kerabat untuk daerah 1: 5
Masukkan 5 nomor rumah kerabat: 3 7 5 8 6
Nomor rumah terurut untuk daerah 1: 3 5 7 8 6
Masukkan jumlah nomor rumah kerabat untuk daerah 2: 4
Masukkan 4 nomor rumah kerabat: 9 7 8 6
Nomor rumah terurut untuk daerah 2: 7 9 8 6
PS C:\Users\HP>

```

Deskripsi Program

Program Go di atas menggunakan selection sort yang bertujuan untuk mengurutkan nomor rumah kerabat berdasarkan kategori ganjil dan genap. Nomor rumah ganjil diurutkan dari yang terkecil ke yang terbesar (menaik) sedangkan nomor rumah genap diurutkan dari yang terbesar ke yang terkecil (menurun). Program ini dimulai dengan meminta jumlah daerah, lalu untuk setiap daerah, program meminta jumlah dan daftar nomor rumah. Nomor rumah dibagi menjadi dua kelompok, yaitu nomor ganjil dan nomor genap. Setelah itu, kedua kelompok ini diurutkan sesuai dengan aturan yang telah ditentukan. Nomor rumah ganjil diurutkan dengan fungsi `urutkanMeningkat` dan nomor rumah genap diurutkan dengan fungsi `urutkanMenurun`. Setelah kedua kelompok terurut, keduanya digabungkan menjadi satu daftar yang terurut.

Fungsi utama dalam program ini adalah `urutkanMeningkat` dan `urutkanMenurun`. Fungsi `urutkanMeningkat` mengurutkan nomor rumah ganjil secara menaik dengan cara mencari elemen terkecil di dalam subarray dan menukarnya dengan elemen pertama. Sedangkan fungsi `urutkanMenurun` mengurutkan nomor rumah genap secara menurun dengan cara mencari elemen terbesar dan menukarnya dengan elemen pertama. Setelah kedua kelompok terurut, hasilnya digabungkan

menggunakan fungsi `append` untuk membentuk daftar akhir yang sudah terurut.

2. Soal Studi Case (Selection Sort)

Kompetisi pemrograman yang baru saja berlalu diikuti oleh 17 tim dari berbagai perguruan tinggi ternama. Dalam kompetisi tersebut, setiap tim berlomba untuk menyelesaikan sebnayak mungkin problem yang diberikan. Dari 13 problem yang diberikan, ada satu problem yang menarik. Problem tersebut mudah dipahami, hampir semua tim mencoba untuk menyelesaikannya, tetapi hanya 3 tim yang berhasil. Apa sih problemnya?

"Median adalah nilai tengah dari suatu koleksi data yang terurut. Jika jumlah data genap, maka nilai median adalah rerata dari kedua nilai tengahnya. Pada problem ini, semua data merupakan bilangan bulat positif dan karenanya rerata nilai tengah dibulatkan ke bawah."

Buatlah program **median** yang mencetak nilai median terhadap seluruh data yang sudah terbaca, jika data yang dibaca saat itu adalah 0.

Masukan berbentuk rangkaian bilangan bulat. Masukan tidak akan berisi lebih dari 1000000 data, tidak termasuk bilangan 0. Data 0 merupakan tanda bahwa median harus dicetak, tidak termasuk data yang dicari mediannya. Data masukan diakhiri dengan bilangan bulat -5313.

Keluaran adalah median yang diminta, satu data per baris.

No	Masukan	Keluaran
1	7 23 11 0 5 19 2 29 3 13 17 0 -5313	11 12

Keterangan:

Sampai bilangan 0 yang pertama, data terbaca adalah 7 23 11, setelah tersusun: 7 11 23, maka median saat itu adalah 11.

Sampai bilangan 0 yang kedua, data adalah 7 23 11 5 19 2 29 3 13 17, setelah tersusun diperoleh: 2 3 5 7 **11 13** 17 19 23 29. Karena ada 10 data, genap, maka median adalah $(11+13)/2=12$.

Petunjuk:

Untuk setiap data bukan 0 (dan bukan marker -5313541) simpan ke dalam array. Dan setiap kali menemukan bilangan 0, urutkanlah data yang sudah tersimpan dengan menggunakan metode insertion sort dan ambil mediannya.

Sourcecode

```

package main

import (
    "fmt"
)

func urutkanAngka(daftarAngka []int) {
    for i := 1; i < len(daftarAngka); i++ {
        angkaKunci := daftarAngka[i]
        j := i - 1
        for j >= 0 && daftarAngka[j] > angkaKunci {
            daftarAngka[j+1] = daftarAngka[j]
            j--
        }
        daftarAngka[j+1] = angkaKunci
    }
}

func hitungMedian(daftarAngka []int) float64 {
    jumlahData := len(daftarAngka)
    if jumlahData == 0 {
        return 0
    }

    if jumlahData%2 != 0 {
        return float64(daftarAngka[jumlahData/2])
    }

    return float64(daftarAngka[(jumlahData-1)/2]+daftarAngka[jumlahData/2]) / 2.0
}

func main() {
    var koleksiAngka []int
    var inputAngka int

    fmt.Println("Masukkan angka (0 untuk menghitung median dan -5313 untuk mengakhiri):")
    for {
        fmt.Scan(&inputAngka)

        if inputAngka == 0 {

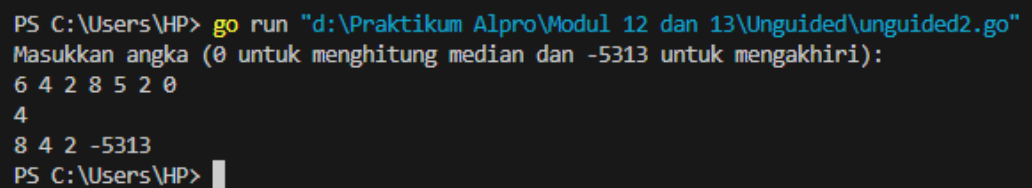
            if len(koleksiAngka) > 0 {
                koleksiSementara := make([]int, len(koleksiAngka))
                copy(koleksiSementara, koleksiAngka)
                urutkanAngka(koleksiSementara)
                median := hitungMedian(koleksiSementara)
                fmt.Printf("%.0f\n", median)
            }
        } else if inputAngka == -5313 {

```

```
        break
    } else {

        koleksiAngka = append(koleksiAngka,
inputAngka)
    }
}
```

Screenshoot Output



```
PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Unguided\unguided2.go"
Masukkan angka (0 untuk menghitung median dan -5313 untuk mengakhiri):
6 4 2 8 5 2 0
4
8 4 2 -5313
PS C:\Users\HP> |
```

Deskripsi Program

Program di atas bertujuan untuk menerima input angka dari pengguna, mengurutkan angka-angka tersebut menggunakan metode insertion sort, lalu menghitung mediannya. Pengguna dapat terus memasukkan angka, dan setiap kali memasukkan angka `0`, program akan menampilkan nilai median dari kumpulan angka yang telah dimasukkan sejauh itu. Jika pengguna memasukkan angka `-5313`, program akan berhenti. Metode insertion sort digunakan dalam fungsi `urutkanAngka` untuk mengurutkan array. Prosesnya melibatkan penyisipan elemen ke dalam posisi yang sesuai dalam array yang sudah diurutkan sebelumnya. Hal ini dilakukan dengan cara membandingkan elemen saat ini dengan elemen sebelumnya dan memindahkan elemen hingga ditemukan posisi yang tepat.

Fungsi penting lainnya adalah `hitungMedian`, yang menghitung median dari kumpulan angka yang telah diurutkan. Median adalah nilai tengah dari data yang diurutkan. Jika jumlah elemen ganjil, median adalah elemen tengah. Namun, jika jumlah elemen genap, median dihitung sebagai rata-rata dari dua elemen tengah. Kedua fungsi ini bekerja sama untuk menerima input angka dari pengguna, mengurutkannya, dan menghitung median setiap kali pengguna meminta hasil dengan memasukkan angka `0`.

3. Soal Studi Case (Insertion Sort)

Sebuah program perpustakaan digunakan untuk mengelola data buku di dalam suatu perpustakaan. Misalnya terdefinisi struct dan array seperti berikut ini:

```
const nMax : integer = 7919
type Buku = <
    id, judul, penulis, penerbit : string
    eksemplar, tahun, rating : integer >

type DaftarBuku = array [ 1..nMax ] of Buku
Pustaka : DaftarBuku
nPustaka: integer
```

Masukan terdiri dari beberapa baris. Baris pertama adalah bilangan bulat N yang menyatakan banyaknya data buku yang ada di dalam perpustakaan. N baris berikutnya, masing-masingnya adalah data buku sesuai dengan atribut atau field pada struct. Baris terakhir adalah bilangan bulat yang menyatakan rating buku yang akan dicari.

Keluaran terdiri dari beberapa baris. Baris pertama adalah data buku terfavorit, baris kedua adalah lima judul buku dengan rating tertinggi, selanjutnya baris terakhir adalah data buku yang dicari sesuai rating yang diberikan pada masukan baris terakhir.

Lengkapi subprogra,-subprogram dibawah ini, sesuai dengan I.S. dan F.S yang diberikan.

```

procedure DaftarkanBuku(in/out pustaka : DaftarBuku, n : integer)
{I.S. sejumlah n data buku telah siap para piranti masukan
 F.S. n berisi sebuah nilai, dan pustaka berisi sejumlah n data buku}

procedure CetakTerfavorit(in pustaka : DaftarBuku, in n : integer)
{I.S. array pustaka berisi n buah data buku dan belum terurut
 F.S. Tampilan data buku (judul, penulis, penerbit, tahun)
terfavorit, yaitu memiliki rating tertinggi}

procedure UrutBuku( in/out pustaka : DaftarBuku, n : integer )
{I.S. Array pustaka berisi n data buku
 F.S. Array pustaka terurut menurun/mengecil terhadap rating.
Catatan: Gunakan metoda Insertion sort}

procedure Cetak5Terbaru( in pustaka : DaftarBuku, n integer )
{I.S. pustaka berisi n data buku yang sudah terurut menurut rating
 F.S. Laporan 5 judul buku dengan rating tertinggi
Catatan: Isi pustaka mungkin saja kurang dari 5}

procedure CariBuku(in pustaka : DaftarBuku, n : integer, r : integer )
{I.S. pustaka berisi n data buku yang sudah terurut menurut rating
 F.S. Laporan salah satu buku (judul, penulis, penerbit, tahun,
eksemplar, rating) dengan rating yang diberikan. Jika tidak ada buku
dengan rating yang ditanyakan, cukup tuliskan "Tidak ada buku dengan
rating seperti itu". Catatan: Gunakan pencarian biner/belah dua.}

```

Sourcecode

```

package main

import "fmt"

const kapasitasMaksimal = 7919

type Buku struct {
    kode, judul, penulis, penerbit string
    stok, tahun, rating                int
}

type DaftarBuku []Buku

func tambahBukuBaru(koleksi *DaftarBuku, total *int) {
    var bukuBaru Buku

    fmt.Print("Masukkan Kode Buku: ")
    fmt.Scan(&bukuBaru.kode)
    fmt.Print("Masukkan Judul: ")
    fmt.Scan(&bukuBaru.judul)
    fmt.Print("Masukkan Penulis: ")
    fmt.Scan(&bukuBaru.penulis)
    fmt.Print("Masukkan Penerbit: ")
    fmt.Scan(&bukuBaru.penerbit)
    fmt.Print("Masukkan Jumlah Stok: ")
}

```



```

        fmt.Scan(&bukuBaru.stok)
        fmt.Print("Masukkan Tahun: ")
        fmt.Scan(&bukuBaru.tahun)
        fmt.Print("Masukkan Rating: ")
        fmt.Scan(&bukuBaru.rating)

        *koleksi = append(*koleksi, bukuBaru)
        *total++
    }

func tampilkanBukuFavorit(koleksi DaftarBuku, total int)
{
    if total == 0 {
        fmt.Println("Tidak ada buku dalam koleksi")
        return
    }

    indeksTerbaik := 0
    for i := 1; i < total; i++ {
        if koleksi[i].rating >
koleksi[indeksTerbaik].rating {
            indeksTerbaik = i
        }
    }

    fmt.Println("\nBuku Favorit:")
    fmt.Printf("Judul: %s\nPenulis: %s\nPenerbit:
%s\nTahun: %d\nRating: %d\n",
        koleksi[indeksTerbaik].judul,
koleksi[indeksTerbaik].penulis,
koleksi[indeksTerbaik].penerbit,
koleksi[indeksTerbaik].tahun,
koleksi[indeksTerbaik].rating)
}

func urutkanBukuRating(koleksi *DaftarBuku, total int) {
    for i := 1; i < total; i++ {
        kunci := (*koleksi)[i]
        j := i - 1
        for j >= 0 && (*koleksi)[j].rating <
kunci.rating {
            (*koleksi)[j+1] = (*koleksi)[j]
            j--
        }
        (*koleksi)[j+1] = kunci
    }
}

func tampilkan5BukuTerbaik(koleksi DaftarBuku, total
int) {
    fmt.Println("\n5 Buku dengan Rating Tertinggi:")
    batas := 5

```

```

        if total < 5 {
            batas = total
        }
        for i := 0; i < batas; i++ {
            fmt.Printf("%d. %s (Rating: %d)\n", i+1,
                koleksi[i].judul, koleksi[i].rating)
        }
    }

    func cariBukuRating(koleksi DaftarBuku, total int,
        ratingDicari int) {
        kiri, kanan := 0, total-1
        ditemukan := false

        for kiri <= kanan {
            tengah := (kiri + kanan) / 2
            if koleksi[tengah].rating == ratingDicari {
                fmt.Printf("\nBuku dengan rating %d
                    ditemukan:\n", ratingDicari)
                fmt.Printf("Judul: %s\nPenulis:
                    %s\nPenerbit: %s\nTahun: %d\nStok: %d\n",
                        koleksi[tengah].judul,
                        koleksi[tengah].penulis,
                        koleksi[tengah].penerbit,
                        koleksi[tengah].tahun,
                        koleksi[tengah].stok)
                ditemukan = true
                break
            }
            if koleksi[tengah].rating < ratingDicari {
                kanan = tengah - 1
            } else {
                kiri = tengah + 1
            }
        }

        if !ditemukan {
            fmt.Printf("\nTidak ada buku dengan rating
                %d\n", ratingDicari)
        }
    }

    func main() {
        var koleksiBuku DaftarBuku
        var totalBuku int
        var pilihanMenu, ratingDicari int

        for {
            fmt.Println("\n=== Sistem Manajemen Buku ===")
            fmt.Println("1. Tambah Buku Baru")
            fmt.Println("2. Tampilkan Buku Favorit")

```

```

        fmt.Println("3. Urutkan Buku berdasarkan
Rating")
        fmt.Println("4. Tampilkan 5 Buku Terbaik")
        fmt.Println("5. Cari Buku berdasarkan Rating")
        fmt.Println("6. Keluar")
        fmt.Print("Pilih menu (1-6): ")
        fmt.Scan(&pilihanMenu)

        switch pilihanMenu {
        case 1:
            tambahBukuBaru(&koleksiBuku, &totalBuku)
        case 2:
            tampilkanBukuFavorit(koleksiBuku, totalBuku)
        case 3:
            urutkanBukuRating(&koleksiBuku, totalBuku)
            fmt.Println("Buku telah diurutkan
berdasarkan rating")
        case 4:
            tampilkan5BukuTerbaik(koleksiBuku,
totalBuku)
        case 5:
            fmt.Print("Masukkan rating yang dicari: ")
            fmt.Scan(&ratingDicari)
            cariBukuRating(koleksiBuku, totalBuku,
ratingDicari)
        case 6:
            fmt.Println("Terima kasih telah menggunakan
sistem manajemen buku")
            return
        default:
            fmt.Println("Pilihan tidak valid")
        }
    }
}

```

Screenshoot Output

```
PS C:\Users\HP> go run "d:\Praktikum Alpro\Modul 12 dan 13\Unguided\unguided3.go"

=== Sistem Manajemen Buku ===
1. Tambah Buku Baru
2. Tampilkan Buku Favorit
3. Urutkan Buku berdasarkan Rating
4. Tampilkan 5 Buku Terbaik
5. Cari Buku berdasarkan Rating
6. Keluar
Pilih menu (1-6): 1
Masukkan Kode Buku: T004
Masukkan Judul: Sorting Hari Ini
Masukkan Penulis: Masukkan Penerbit: Masukkan Jumlah Stok: 60
Masukkan Tahun: 2024
Masukkan Rating: 5

=== Sistem Manajemen Buku ===
1. Tambah Buku Baru
2. Tampilkan Buku Favorit
3. Urutkan Buku berdasarkan Rating
4. Tampilkan 5 Buku Terbaik
5. Cari Buku berdasarkan Rating
6. Keluar
Pilih menu (1-6): █
```

Deskripsi Program

Program di atas adalah sistem manajemen buku yang memungkinkan pengguna untuk menambahkan, mengelola, dan mencari informasi tentang koleksi buku berdasarkan rating. Salah satu fitur penting dalam program ini adalah kemampuan untuk mengurutkan koleksi buku berdasarkan rating menggunakan algoritma insertion sort. Pengurutan ini dilakukan oleh fungsi `urutkanBukuRating`, yang menyusun buku-buku dari rating tertinggi ke terendah. Hal ini mempermudah pengguna untuk melihat buku dengan rating terbaik secara berurutan, misalnya saat menampilkan lima buku teratas. Program juga menyediakan opsi untuk menampilkan buku favorit dengan rating tertinggi, menambahkan buku baru, dan mencari buku berdasarkan rating tertentu menggunakan pencarian biner.

Fungsi `urutkanBukuRating` adalah salah satu bagian penting dari program ini. Fungsi ini menggunakan metode insertion sort, dimana setiap elemen koleksi dibandingkan dengan elemen-elemen sebelumnya, lalu dipindahkan ke posisi yang tepat agar urutan tetap terjaga. Proses ini dilakukan dengan perulangan yang memastikan koleksi terurut hingga elemen terakhir.

IV. DAFTAR PUSTAKA

- Munir, R. (2009). Struktur Diskrit: Makalah Kuliah Matematika Diskrit. Diakses pada 1 Desember 2024, dari <https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2009-2010/Makalah0910/MakalahStrukdis0910-074.pdf>.
- TutorialsPoint. (n.d.). Selection sort in Go lang. Diakses pada 1 Desember 2024, dari <https://www.tutorialspoint.com/selection-sort-in-go-lang>
- Wayan. (n.d.). Struktur data: Pertemuan 6. Universitas Pembangunan Jaya. Diakses pada 1 Desember 2024, dari <https://ocw.upj.ac.id/files/Handout-INF202-INF202-Struktur-Data-Wayan-Pertemuan-6.pdf>