LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 12 & 13

PENGURUTAN DASAR

Disusun Oleh :

Agnes Refilina Fiska / 2311102126

S1-IF-11-05

Dosen Pengampu :

Arif Amrulloh, S.Kom.,M.Kom

PROGRAM STUDI S1 TEKNIK

INFORMATIKA FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO 2024

I. DASAR TEORI

A. Penjelasan Pengurutan Data

Pengurutan data merupakan proses menyusun elemen-elemen dalam sebuah daftar sesuai dengan urutan tertentu, seperti dari yang terkecil ke terbesar atau sebaliknya. Metode ini sering digunakan di berbagai bidang untuk mempercepat pencarian, analisis data, serta pemrosesan lainnya.

1. Ide Algoritma Selection Sort

Selection Sort adalah salah satu algoritma pengurutan yang bekerja dengan cara mencari nilai ekstrem (terkecil atau terbesar) dari sekumpulan data yang belum diurutkan, kemudian menempatkannya pada posisi yang sesuai. Misalnya, jika pengurutan dilakukan secara menaik (ascending), algoritma akan mencari nilai terkecil dari data yang tersisa, lalu menukarnya dengan data yang berada pada posisi paling kiri dari rentang tersebut. Proses ini diulang secara bertahap dengan mempersempit rentang data yang belum diurutkan hingga hanya tersisa satu elemen. Dalam Selection Sort, ada dua langkah utama, yaitu mencari nilai ekstrem dan melakukan pertukaran (swap) posisi dengan elemen tertentu. Algoritma ini memiliki keunggulan dalam kesederhanaannya, tetapi kurang efisien untuk dataset yang besar karena kompleksitas waktu yang dimilikinya adalah O(n2)O(n^2).

2. Algoritma Selection Sort

Algoritma Selection Sort bekerja dengan langkah-langkah berikut: Pertama, mulai dari indeks awal, carilah elemen terkecil dari bagian daftar yang belum diurutkan. Setelah itu, tukarkan elemen terkecil tersebut dengan elemen yang berada di posisi awal dari bagian yang belum terurut. Selanjutnya, geser batas daftar yang belum diurutkan ke kanan sebanyak satu elemen. Proses ini diulangi hingga seluruh elemen dalam daftar telah tersusun rapi. Kompleksitas waktu algoritma ini, baik dalam kasus terbaik, rata-rata, maupun terburuk, adalah O(n2)O(n^2), di mana nn adalah jumlah elemen dalam daftar. Meskipun algoritma ini sederhana untuk diimplementasikan, namun tidak efisien untuk digunakan pada daftar dengan ukuran besar.

	Notasi Algoritma	Notasi dalam bahasa Go
1	i ← 1	i = 1
2	while i <= n-1	for i <= n-1 {
3	idx_min ← i - 1	idx_min = i - 1
4	j ← i	j = i
5	while j < n do	for j < n {
6	if a[idx_min] > a[j] then	if a[idx_min] > a[j] {
7	idx_min ← j	idx_min = j
8	endif	}
9	j ← j + 1	j = j + 1
10	endwhile	}
11	t ← a[idx_min]	t = a[idx_min]
12	a[idx_min] ← a[i-i]	a[idx_min] = a[i-1]
13	a[i-1] ← t	a[i-1] = t
14	i ← i + 1	i = i + 1
15	endwhile	}

3. Ide Algoritma Insertion Sort

Insertion Sort adalah salah satu algoritma pengurutan yang bekerja dengan cara menyisipkan elemen ke dalam posisi yang tepat pada bagian daftar yang sudah terurut. Tidak seperti Selection Sort yang mencari nilai ekstrem terlebih dahulu, Insertion Sort memproses elemen satu per satu dengan membandingkan elemen yang belum terurut dengan elemen-elemen di bagian yang sudah terurut. Jika diperlukan, elemen-elemen yang sudah terurut akan digeser ke kanan untuk memberikan ruang bagi elemen yang sedang diproses agar dapat disisipkan ke posisi yang sesuai. Proses ini diulang untuk setiap elemen yang belum terurut hingga seluruh elemen berada dalam urutan yang benar. Dalam penjelasan pada gambar, data diurutkan secara menurun (descending), sehingga elemen terbesar berada di sisi kiri. Algoritma ini sederhana dan efisien untuk daftar kecil atau yang hampir terurut, tetapi memiliki kompleksitas waktu O(n2)O(n^2) dalam kasus rata-rata dan terburuk, sehingga kurang cocok untuk daftar berukuran besar.

	Notasi Algoritma	Notasi dalam bahasa Go
1	i ← 1	i = 1
2	while i <= n-1	r i <= n-1 {
3	j ← i	j = i
4	temp ← a[j]	temp = a[j]
5	while j > 0 and > a[j-1] do	for j > 0 && > a[j-1] {
6	a[j] ← a[j-1]	a[j] = a[j-1]
7	j ← j - 1	j = j - 1
8	endwhile	}
9	a[j] ← temp	a[j] = temp
i0	i ← i + i	i = i + i
11	endwhile	}

4. Algoritma Insertion Sort

Algoritma Insertion Sort memiliki langkah-langkah sebagai berikut: Dimulai dari elemen kedua dalam daftar, dengan asumsi bahwa elemen pertama sudah dalam keadaan terurut. Selanjutnya, elemen tersebut dibandingkan dengan elemen-elemen dalam bagian daftar

yang sudah terurut. Jika ada elemen yang lebih besar, geser elemen-elemen tersebut ke kanan untuk memberikan ruang bagi elemen yang sedang diproses. Setelah itu, masukkan elemen tersebut ke posisi yang sesuai dalam bagian yang sudah terurut. Langkah ini diulangi untuk setiap elemen dalam daftar hingga seluruhnya terurut. Kompleksitas waktu algoritma ini adalah O(n)O(n) pada kasus terbaik (ketika daftar hampir terurut) dan O(n2)O(n^2) pada kasus terburuk (ketika daftar terbalik). Algoritma ini sangat cocok untuk daftar berukuran kecil atau daftar yang hampir terurut.

II. GUIDED

1. Guided 1

Soal Studi Case

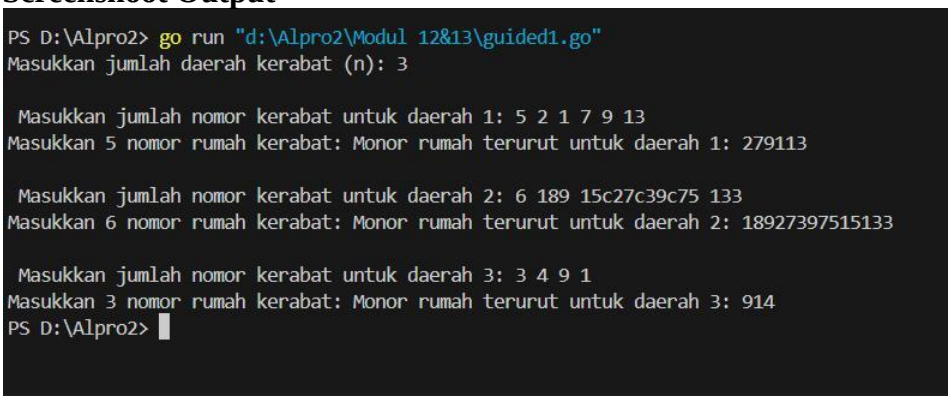
Pengurutan nomor rumah kerabat dengan selection sort

Sourcecode

```
package main

import (
    "fmt"
)

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

        }
       arr[i], arr[idxMin] = arr[idxMin], arr[i]
    }
}

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

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

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

          fmt.Printf("Monor rumah terurut untuk
daerah %d: ", daerah)
```

```
            for _, num := range arr {
                fmt.Printf("%d", num)
            }
            fmt.Println()
        }

    }
```

Screenshoot Output



Deskripsi Program :

Program di atas adalah implementasi algoritma **Selection Sort** dalam bahasa Go untuk mengurutkan nomor rumah kerabat berdasarkan daerah. Program diawali dengan meminta pengguna untuk memasukkan jumlah daerah kerabat (\(n\)). Untuk setiap daerah, pengguna akan diminta memasukkan jumlah nomor rumah (\(m\)) dan daftar nomor rumah yang akan diurutkan.

Setelah menerima input, program menggunakan fungsi `selectionSort` untuk mengurutkan nomor rumah. Fungsi ini bekerja dengan memilih elemen terkecil dari daftar yang belum terurut dan menukarnya dengan elemen pada posisi yang sesuai, mengulang proses ini hingga seluruh elemen terurut. Setelah pengurutan selesai, program menampilkan daftar nomor rumah yang sudah diurutkan untuk setiap daerah.

Program ini menggunakan perulangan untuk memproses setiap daerah secara terpisah, menjadikan input lebih terstruktur berdasarkan wilayah. Output berupa daftar nomor rumah yang terurut untuk setiap daerah. Program ini berguna untuk membantu menyusun data secara terurut, terutama dalam kasus yang melibatkan banyak wilayah dengan daftar elemen yang berbeda-beda.

2. **Guided 2**

   Soal Studi Case

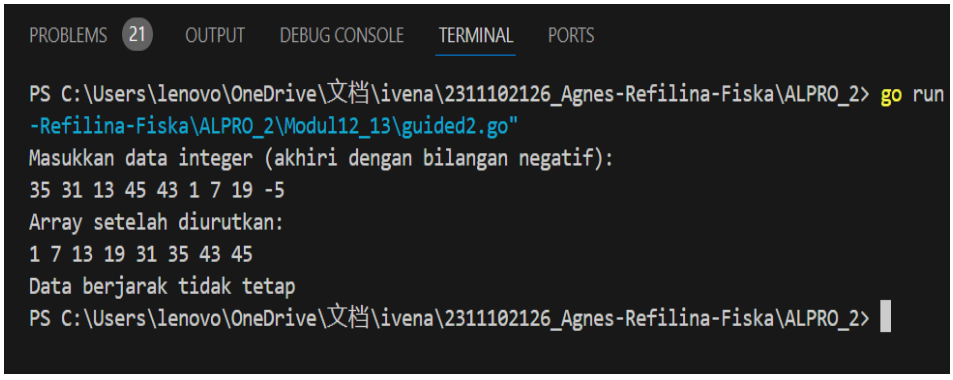
   Pengurutan dan analisis pola selisih elemen array

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



Deskripsi Program :
Program di atas adalah implementasi algoritma Insertion Sort dalam bahasa Go untuk mengurutkan array integer dan memeriksa apakah selisih antara elemen-elemen array tersebut tetap. Program ini dimulai dengan meminta pengguna untuk memasukkan sejumlah bilangan bulat (integer) hingga pengguna memasukkan bilangan negatif sebagai tanda akhir input. Seluruh bilangan yang dimasukkan akan dimasukkan ke dalam array.

Setelah data terkumpul, program menggunakan fungsi `insertionSort` untuk mengurutkan elemen-elemen dalam array. Algoritma Insertion Sort bekerja dengan membandingkan elemen-elemen dalam array secara berurutan, menggeser elemen yang lebih besar ke kanan, dan menyisipkan elemen ke posisi yang sesuai hingga array terurut.

Setelah array diurutkan, program memeriksa apakah selisih antara setiap elemen dalam array tetap menggunakan fungsi `isConstantDifference`. Fungsi ini mengevaluasi perbedaan antara elemen-elemen yang berurutan. Jika selisihnya konsisten, fungsi akan mengembalikan nilai `true` beserta nilai selisih tersebut. Jika tidak, fungsi akan mengembalikan `false`.

Program kemudian menampilkan array yang sudah diurutkan, diikuti dengan informasi mengenai status selisih elemen. Jika selisih elemen tetap, program akan mencetak nilai selisih tersebut. Jika tidak, program akan memberi tahu bahwa selisih elemen tidak tetap. Program ini berguna untuk menganalisis pola data yang memiliki interval tetap atau beraturan.

III. UNGUIDED

1. Unguided 1

Soal Studi Case

Belakangan diketahui ternyata Hercules itu tidak berani menyeberang jalan, maka selalu diusahakan agar hanya menyeberang jalan sesedikit mungkin, hanya diujung 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 normor 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.

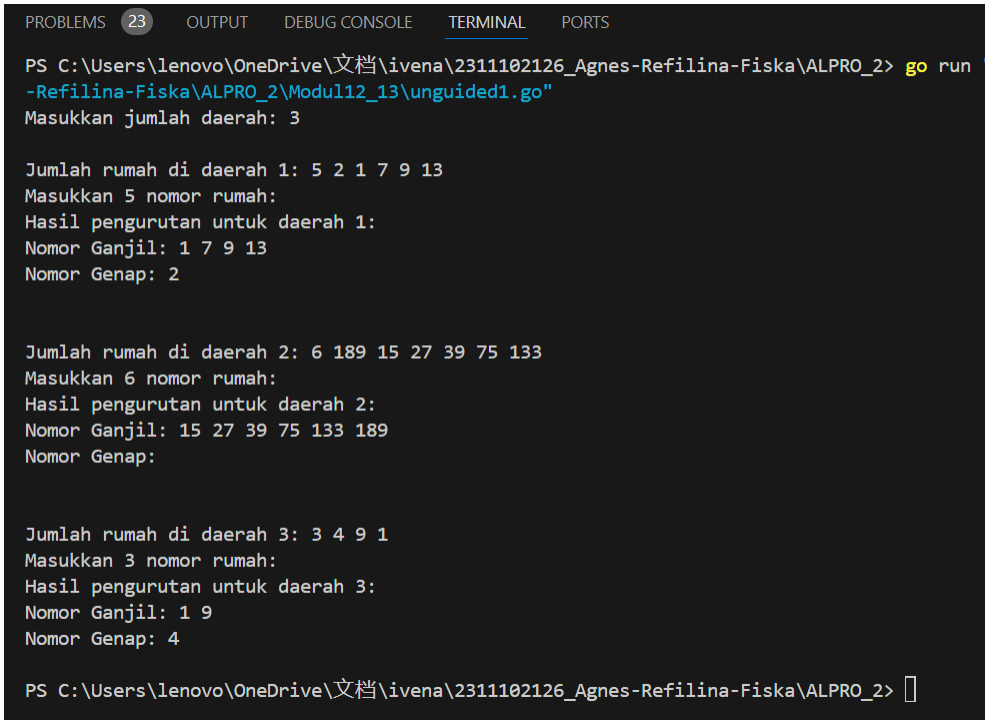
No	Masukan	Keluaran
1	3 5 2 1 7 9 13 6 189 15 27 39 75 133 3 4 9 1	1 13 12 8 2 15 27 39 75 133 189 8 4 2

Sourcecode

```
package main

import (
    "fmt"
    "sort"
)

// Struct untuk menyimpan data daerah
type Daerah struct {
    ganjil []int
    genap  []int
}

// Fungsi untuk memisahkan dan mengurutkan nomor
rumah
func prosesNomorRumah(nomor []int) Daerah {
    var hasil Daerah
    // Pemisahan nomor ganjil dan genap
    for _, n := range nomor {
        if n%2 == 0 {
            hasil.genap = append(hasil.genap, n)
        } else {
            hasil.ganjil = append(hasil.ganjil, n)
        }
    }
    // Pengurutan menggunakan package sort
    sort.Ints(hasil.ganjil)
    // Urutkan ganjil naik

sort.Sort(sort.Reverse(sort.IntSlice(hasil.genap)))
// Urutkan genap turun
    return hasil
}

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

    // Proses untuk setiap daerah
    for i := 1; i <= jumlahDaerah; i++ {
        var jumlahRumah int
        fmt.Printf("\nJumlah rumah di daerah %d: ",
i)
        fmt.Scan(&jumlahRumah)
```

```
        // Membuat slice untuk nomor rumah
        rumah := make([]int, 0, jumlahRumah)
        fmt.Printf("Masukkan %d nomor rumah: ",
jumlahRumah)

        // Input nomor rumah
        for j := 0; j < jumlahRumah; j++ {
            var nomor int
            fmt.Scan(&nomor)
            rumah = append(rumah, nomor)
        }

        // Proses pengurutan
        hasil := prosesNomorRumah(rumah)

        // Output hasil
        fmt.Printf("\nHasil pengurutan untuk daerah
%d:\n", i)
        fmt.Print("Nomor Ganjil: ")
        for _, n := range hasil.ganjil {
            fmt.Printf("%d ", n)
        }
        fmt.Println()
        fmt.Print("Nomor Genap: ")
        for _, n := range hasil.genap {
            fmt.Printf("%d ", n)
        }
        fmt.Println("\n")
    }
}
```

Screenshoot Output



Deskripsi Program :

Program di atas adalah implementasi dalam bahasa Go untuk memisahkan dan mengurutkan nomor-nomor rumah di beberapa daerah. Pengguna akan diminta untuk memasukkan jumlah daerah yang akan diproses, kemudian untuk setiap daerah, pengguna akan diminta untuk memasukkan jumlah rumah dan nomor-nomor rumah tersebut.

Program akan memproses data yang diberikan dengan melakukan pemisahan antara nomor rumah ganjil dan genap. Setelah itu, nomor-nomor rumah ganjil akan diurutkan secara ascending, sedangkan nomor-nomor rumah genap akan diurutkan secara descending. Hasil dari proses ini akan ditampilkan kepada pengguna, dengan pemisahan antara nomor rumah ganjil dan genap.

Dalam program ini, terdapat dua fungsi utama. Fungsi `prosesNomorRumah()` bertugas untuk memisahkan dan mengurutkan nomor-nomor rumah, sedangkan fungsi `main()` bertanggung jawab untuk mengambil input dari pengguna, memanggil fungsi `prosesNomorRumah()`, dan menampilkan output.

Pemilihan algoritma pengurutan pada program ini menggunakan built-in `sort.Ints()` untuk nomor rumah ganjil dan `sort.Sort(sort.Reverse(sort.IntSlice(hasil.genap)))` untuk nomor rumah genap. Hal ini dilakukan karena algoritma tersebut sudah cukup efisien dan mudah diimplementasikan.

Secara keseluruhan, program ini memberikan solusi yang sederhana untuk memisahkan dan mengurutkan nomor-nomor rumah di beberapa daerah, dengan output yang jelas dan mudah dipahami.

2. Unguided 2

Soal Studi Case

Kompetisi pemrograman yang baru saja berlalu diikuti oleh 17 tim dari berbagai perguruan tinggi ternama. Dalam kompetisi tersebut, setiap tim berlomba untuk menyelesaikan sebanyak 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 sudah 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.

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

Sourcecode

```
package main

import (
    "fmt"
    "sort"
)

// Fungsi untuk menghitung median
func findMedian(numbers []int) float64 {
    sort.Ints(numbers)
    n := len(numbers)
    if n == 0 {
        return 0
    }
    if n%2 == 0 {
        return float64(numbers[n/2-1]+numbers[n/2])
/ 2
    } else {
        return float64(numbers[n/2])
    }
```

```
}

func main() {
    var numbers []int
    var input int

    fmt.Println("Enter numbers (0 to calculate
median, -5313 to stop):")

    for {
        fmt.Scan(&input)
        if input == 0 {
            if len(numbers) > 0 {
                median := findMedian(numbers)
                fmt.Printf("Median: %.0f\n",
median)
            }
        } else if input == -5313 {
            break
        } else {
            numbers = append(numbers, input)
        }
    }
}
```

Screenshoot Output

Deskripsi Program :
Program di atas adalah implementasi dalam bahasa Go untuk memisahkan dan mengurutkan nomor rumah berdasarkan sifat bilangan (ganjil atau genap) di beberapa daerah. Program dimulai dengan meminta pengguna untuk memasukkan jumlah daerah. Untuk setiap daerah, pengguna diminta memasukkan jumlah nomor rumah dan daftar nomor rumah tersebut.

Program memproses nomor rumah menggunakan fungsi `prosesNomorRumah`, yang memisahkan bilangan ganjil dan genap ke

dalam dua daftar terpisah. Nomor rumah ganjil diurutkan secara menaik menggunakan fungsi `sort.Ints`, sementara nomor rumah genap diurutkan secara menurun menggunakan kombinasi `sort.Reverse` dan `sort.IntSlice`. Hasil pengurutan disimpan dalam sebuah struct bernama `Daerah` yang memiliki dua atribut, yaitu daftar nomor ganjil dan genap.

Setelah pengurutan selesai, program menampilkan hasilnya untuk setiap daerah. Nomor rumah ganjil ditampilkan lebih dahulu dalam urutan menaik, diikuti oleh nomor rumah genap dalam urutan menurun. Program ini sangat berguna untuk mengorganisasi data dengan kriteria tertentu, seperti pengelompokan berdasarkan sifat bilangan dan pengurutan sesuai kebutuhan.

3. Unguided 3

Soal Studi Case

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 subprogram-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 maxBuku = 7919

// Struct untuk data buku
type Buku struct {
    id        string
    judul     string
    penulis   string
    penerbit  string
    eksemplar int
    tahun     int
    rating    int
}

// Struct untuk daftar buku
type KoleksiBuku []Buku

// Fungsi untuk menambahkan buku baru
func  TambahkanBuku(koleksi  *KoleksiBuku,  jumlah
*int) {
    var bukuBaru Buku
    fmt.Print("Masukkan ID: ")
    fmt.Scan(&bukuBaru.id)
    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 Eksemplar: ")
    fmt.Scan(&bukuBaru.eksemplar)
    fmt.Print("Masukkan Tahun: ")
    fmt.Scan(&bukuBaru.tahun)
    fmt.Print("Masukkan Rating: ")
    fmt.Scan(&bukuBaru.rating)

    *koleksi = append(*koleksi, bukuBaru)
    *jumlah++
```

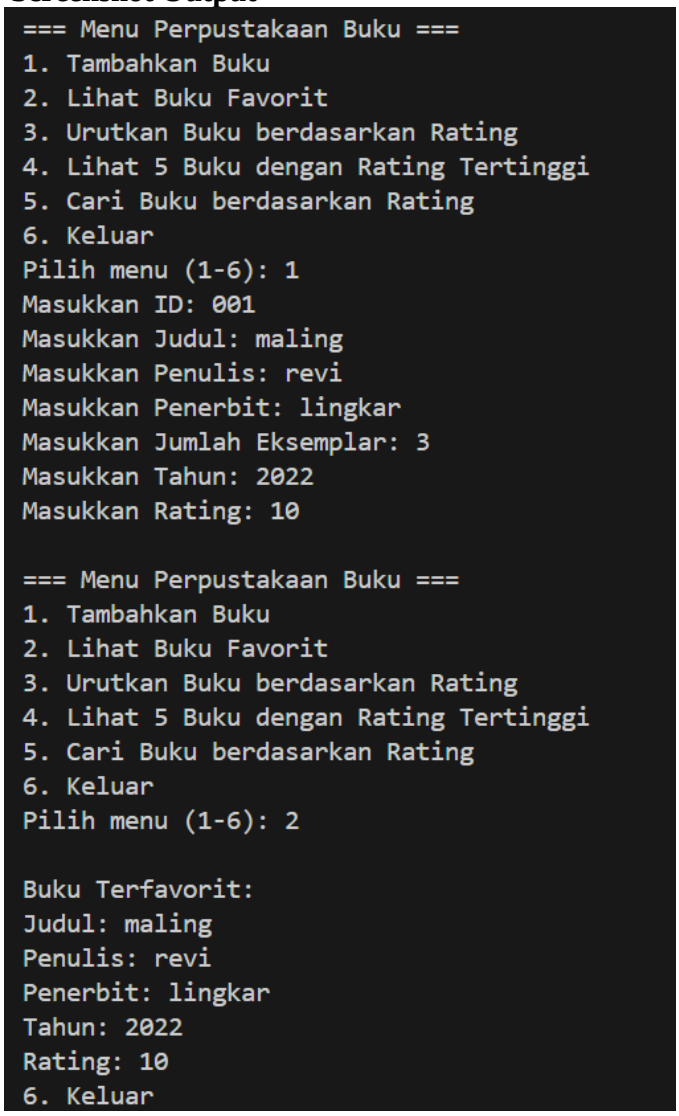
```
}

// Fungsi untuk mencetak buku dengan rating tertinggi
func TampilkanFavorit(koleksi KoleksiBuku, jumlah int) {
    if jumlah == 0 {
            fmt.Println("Tidak ada buku dalam koleksi")
        return
    }

    ratingTertinggi := koleksi[0].rating
    indeksFavorit := 0
    for i := 1; i < jumlah; i++ {
        if koleksi[i].rating > ratingTertinggi {
            ratingTertinggi = koleksi[i].rating
            indeksFavorit = i
        }
    }

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

// Fungsi untuk mengurutkan buku berdasarkan rating menggunakan insertion sort
func SortirBuku(koleksi *KoleksiBuku, jumlah int) {
    for i := 1; i < jumlah; i++ {
        bukuSementara := (*koleksi)[i]
        j := i - 1
           for j >= 0 && (*koleksi)[j].rating < bukuSementara.rating {
            (*koleksi)[j+1] = (*koleksi)[j]
            j--
        }
        (*koleksi)[j+1] = bukuSementara
    }
}
```

```
// Fungsi untuk menampilkan 5 buku dengan rating tertinggi
func CetakTop5(koleksi KoleksiBuku, jumlah int) {
        fmt.Println("\n5 Buku dengan Rating Tertinggi:")
    jumlahTampil := 5
    if jumlah < 5 {
        jumlahTampil = jumlah
    }
    for i := 0; i < jumlahTampil; i++ {
         fmt.Printf("%d. %s (Rating: %d)\n", i+1, koleksi[i].judul, koleksi[i].rating)
    }
}

// Fungsi untuk mencari buku dengan rating tertentu menggunakan binary search
func TemukanBuku(koleksi KoleksiBuku, jumlah int, target int) {
    kiri := 0
    kanan := jumlah - 1
    ditemukan := false

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

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

```
    }
}

func main() {
    var koleksi KoleksiBuku
    var jumlahBuku int
    var pilihan int
    var ratingTarget int

    for {
         fmt.Println("\n=== Menu Perpustakaan Buku
===")
        fmt.Println("1. Tambahkan Buku")
        fmt.Println("2. Lihat Buku Favorit")
         fmt.Println("3. Urutkan Buku berdasarkan
Rating")
        fmt.Println("4. Lihat 5 Buku dengan Rating
Tertinggi")
           fmt.Println("5.  Cari  Buku  berdasarkan
Rating")
        fmt.Println("6. Keluar")
        fmt.Print("Pilih menu (1-6): ")
        fmt.Scan(&pilihan)

        switch pilihan {
        case 1:
            TambahkanBuku(&koleksi, &jumlahBuku)
        case 2:
            TampilkanFavorit(koleksi, jumlahBuku)
        case 3:
            SortirBuku(&koleksi, jumlahBuku)
              fmt.Println("Buku berhasil diurutkan
berdasarkan rating")
        case 4:
            CetakTop5(koleksi, jumlahBuku)
        case 5:
             fmt.Print("Masukkan rating yang ingin
dicari: ")
            fmt.Scan(&ratingTarget)
                 TemukanBuku(koleksi,  jumlahBuku,
ratingTarget)
        case 6:
                 fmt.Println("Terima  kasih  telah
menggunakan program koleksi buku.")
            return
        default:
            fmt.Println("Pilihan tidak valid. Coba
lagi.")
```

```
            }
        }
}
```

Screenshot Output



Deskripsi Program :

Program di atas adalah aplikasi berbasis konsol untuk mengelola koleksi buku di perpustakaan. Aplikasi ini memungkinkan pengguna untuk menambahkan data buku, mencetak buku dengan rating tertinggi, mengurutkan koleksi berdasarkan rating, menampilkan lima buku dengan rating terbaik, serta mencari buku tertentu berdasarkan rating menggunakan algoritma pencarian biner. Setiap buku dalam koleksi disimpan sebagai elemen dalam slice bertipe KoleksiBuku, yang terdiri dari struktur data Buku dengan atribut seperti ID, judul, penulis, penerbit, jumlah eksemplar, tahun penerbitan, dan rating.

Pengguna dapat mengakses berbagai fungsi melalui menu utama, termasuk: menambahkan buku baru ke dalam koleksi, melihat buku dengan rating tertinggi, mengurutkan koleksi secara menurun berdasarkan

rating menggunakan algoritma *insertion sort*, mencetak daftar lima buku terbaik, dan mencari buku dengan rating tertentu. Program ini dirancang untuk memberikan pengalaman manajemen koleksi buku yang efisien dengan antarmuka sederhana.

IV. KESIMPULAN

Dari laporan praktikum tentang pengurutan dasar menggunakan algoritma Selection Sort dan Insertion Sort, dapat disimpulkan bahwa kedua algoritma ini memiliki kelebihan dan kekurangan masing-masing.

Selection Sort bekerja dengan cara mencari elemen terkecil dalam daftar yang belum terurut dan menukarnya dengan elemen yang ada pada posisi awal bagian tersebut. Proses ini diulang hingga seluruh daftar terurut. Algoritma ini mudah diimplementasikan, namun memiliki kompleksitas waktu $O(n^2)$ sehingga kurang efisien untuk dataset yang besar.

Di sisi lain, Insertion Sort bekerja dengan cara menyisipkan elemen ke dalam posisi yang tepat pada bagian yang sudah terurut. Setiap elemen dibandingkan dengan elemen-elemen sebelumnya dan dipindahkan ke posisi yang sesuai. Insertion Sort lebih efisien daripada Selection Sort pada data yang sudah hampir terurut, dengan kompleksitas waktu $O(n)$ pada kasus terbaik. Namun, pada kasus terburuk, Insertion Sort tetap memiliki kompleksitas $O(n^2)$.

Secara umum, pemilihan algoritma pengurutan yang tepat harus mempertimbangkan karakteristik data yang akan diurutkan dan kebutuhan aplikasi yang bersangkutan. Meskipun Selection Sort dan Insertion Sort memiliki keterbatasan dalam efisiensi untuk dataset besar, keduanya tetap berguna dalam pengurutan data kecil atau data yang hampir terurut.

V. REFERENSI

[1] Modul 12 & 13 Praktikum Algoritma 2