

**LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2**

**MODUL XII & XIII
PENGURUTAN DATA**



Disusun Oleh :

Nia Novela Ariandini / 2311102057

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

Selection Sort dan Insertion Sort adalah dua algoritma pengurutan dasar yang sering digunakan untuk memahami konsep dasar sorting dalam pemrograman. Keduanya memiliki cara kerja dan karakteristik yang berbeda, namun sama-sama digunakan untuk mengurutkan data dalam array atau list.

1. Selection Sort

Selection Sort adalah salah satu metode pengurutan sederhana yang bekerja dengan cara mencari elemen terkecil dalam sebuah array yang belum diurutkan dan menukarnya dengan elemen pertama pada iterasi tersebut. Algoritma ini memiliki mekanisme kerja yang sistematis dan mudah dipahami, di mana array dibagi menjadi dua bagian: bagian terurut dan bagian belum terurut. Pada setiap iterasi, algoritma akan melakukan pencarian elemen terkecil di antara elemen-elemen yang belum terurut, kemudian menukarnya dengan elemen pertama pada bagian belum terurut. Proses ini berlanjut secara berulang hingga seluruh elemen array terurut dari nilai terkecil hingga terbesar.

2. Insertion Sort

Insertion Sort adalah algoritma pengurutan yang bekerja dengan cara membandingkan dan menempatkan setiap elemen pada posisi yang tepat di antara elemen-elemen yang telah diurutkan sebelumnya. Algoritma ini membangun array terurut secara bertahap dengan cara menyisipkan setiap elemen baru ke dalam posisi yang benar di antara elemen-elemen yang telah diurutkan. Mekanisme kerjanya dimulai dari elemen kedua dalam array, di mana algoritma membandingkan elemen tersebut dengan elemen-elemen sebelumnya dan memindahkannya ke posisi yang tepat. Proses ini berlanjut untuk setiap elemen berikutnya hingga seluruh array terurut dengan sempurna. Keunggulan Insertion Sort terletak pada efisiensinya untuk array berukuran kecil atau array yang sudah hampir terurut.

II. GUIDED

1. GUIDED 1

Studi Case : Pengurutan nomor rumah kerabat dengan selection sort.

Sourcecode

```
package main

import "fmt"

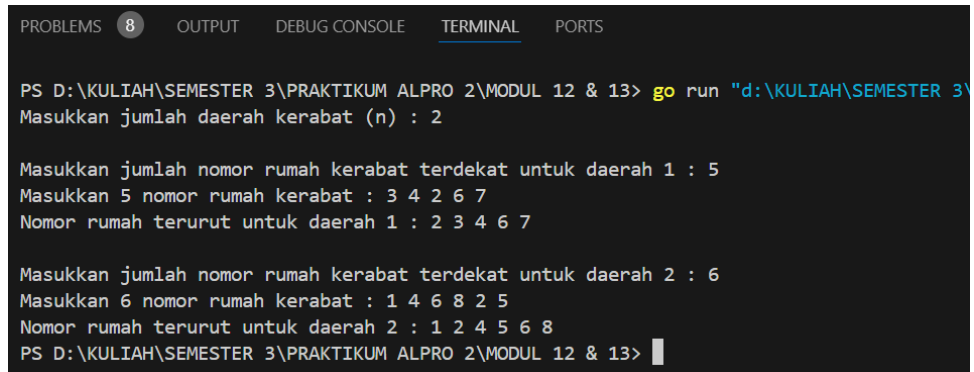
func selectionSort(arr []int, n int) {
    for i := 0; i < n-1; i++ {
        idxMin := i
        for j := i + 1; j < n; j++ {
            //cari elemen terkecil
            if arr[j] < arr[idxMin] {
                idxMin = j
            }
        }
        //tukar elemen terkecil dengan elemen di posisi 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
        terdekat 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



```
PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> go run "d:\KULIAH\SEMESTER 3\  
Masukkan jumlah daerah kerabat (n) : 2  
  
Masukkan jumlah nomor rumah kerabat terdekat untuk daerah 1 : 5  
Masukkan 5 nomor rumah kerabat : 3 4 2 6 7  
Nomor rumah terurut untuk daerah 1 : 2 3 4 6 7  
  
Masukkan jumlah nomor rumah kerabat terdekat untuk daerah 2 : 6  
Masukkan 6 nomor rumah kerabat : 1 4 6 8 2 5  
Nomor rumah terurut untuk daerah 2 : 1 2 4 5 6 8  
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> |
```

Deskripsi Program

Program di atas bertujuan untuk mengurutkan nomor rumah kerabat di beberapa daerah menggunakan algoritma Selection Sort. Program dimulai dengan meminta pengguna untuk memasukkan jumlah daerah yang akan diproses. Setelah itu, untuk setiap daerah, program meminta pengguna untuk memasukkan jumlah nomor rumah kerabat yang ada di daerah tersebut. Nomor rumah yang dimasukkan kemudian disimpan dalam sebuah array.

Algoritma dan Cara Program Berfungsi : Pertama program mengurutkan array menggunakan algoritma Selection Sort. Algoritma ini bekerja dengan cara memilih elemen terkecil dalam array dan menukarnya dengan elemen pada posisi yang sedang diproses. Proses ini dilakukan secara berulang mulai dari elemen pertama hingga elemen terakhir. Dalam setiap iterasi, elemen terkecil dari sisa array akan dipindahkan ke posisi yang sesuai. Proses ini diulang untuk setiap elemen hingga seluruh array terurut.

2. GUIDED 2

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.Print("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.Print("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

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> go run "d:\KULIAH\SEMESTER 3\
Masukkan data integer (akhiri dengan bilangan negatif) : 1 3 5 7 9 11 -13
Array setelah diurutkan : 1 3 5 7 9 11
Data berjarak 2
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13>

```

Deskripsi Program

Program di atas adalah implementasi untuk mengurutkan array menggunakan algoritma Insertion Sort dan memeriksa apakah selisih antara elemen-elemen array yang telah diurutkan bersifat tetap (konstan). Program ini dimulai dengan meminta pengguna untuk memasukkan serangkaian bilangan bulat (integer). Proses input data berakhir jika pengguna memasukkan bilangan negatif, yang menandakan akhir dari data.

Algoritma dan Cara Program Berfungsi : Pertama program ini meminta pengguna untuk memasukkan bilangan bulat secara berurutan, dan proses input berakhir saat bilangan negatif dimasukkan. Setelah itu, program mengurutkan bilangan menggunakan Insertion Sort, di mana elemen dimasukkan ke posisi yang sesuai dalam subarray yang telah diurutkan. Setelah proses pengurutan selesai, program memeriksa apakah selisih antara setiap elemen berturut-turut dalam array adalah tetap (konstan). Pemeriksaan ini dilakukan dengan menghitung selisih dua elemen pertama sebagai referensi, kemudian membandingkan selisih ini dengan setiap pasangan elemen lainnya. Jika selisih tetap ditemukan, program akan menampilkan array yang telah diurutkan beserta nilai selisihnya. Sebaliknya, jika tidak ada selisih tetap, program menyatakan bahwa data tidak memiliki jarak tetap. Program ini sederhana namun efektif untuk memeriksa pola aritmetika dalam data, dan penggunaan Insertion Sort menjadikannya cocok untuk jumlah data yang kecil hingga sedang karena algoritma ini mudah dipahami dan diimplementasikan.

III. UNGUIDED

1. UNGUIDED 1

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"
)

func main() {
    // Masukkan jumlah daerah
    var t int
    fmt.Print("Masukkan Jumlah Daerah (n) : ")
    fmt.Scan(&t)

    for i := 1; i <= t; i++ {
        // Input jumlah rumah untuk daerah ke-i
        var n int
        fmt.Printf("\nMasukkan Jumlah Rumah Daerah %d : ", i)
        fmt.Scan(&n)

        // Input nomor rumah untuk daerah ke-i
        rumah := make([]int, n)
        fmt.Printf("Masukkan %d Nomor Rumah Daerah %d : ", n, i)
        for j := 0; j < n; j++ {
            fmt.Scan(&rumah[j])
        }
    }
}
```

```

        // Memisahkan nomor ganjil dan genap
        var ganjil, genap []int
        for _, nomor := range rumah {
            if nomor%2 == 0 {
                genap = append(genap, nomor)
            } else {
                ganjil = append(ganjil, nomor)
            }
        }

        // Mengurutkan ganjil membesar dan genap mengecil
        sort.Ints(ganjil)
        sort.Sort(sort.Reverse(sort.IntSlice(genap)))

        // Menampilkan hasil dalam format seperti gambar
        fmt.Printf("Nomor Rumah Terurut (Ganjil - Genap)
Daerah %d : ", i)
        for _, nomor := range ganjil {
            fmt.Print(nomor, " ")
        }
        for _, nomor := range genap {
            fmt.Print(nomor, " ")
        }
        fmt.Println()
    }
}

```

Screenshoot Output

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> go run "d:\KULIAH\SEMESTER 3\
Masukkan Jumlah Daerah (n) : 2

Masukkan Jumlah Rumah Daerah 1 : 5
Masukkan 5 Nomor Rumah Daerah 1 : 1 2 3 4 5
Nomor Rumah Terurut (Ganjil - Genap) Daerah 1 : 1 3 5 4 2

Masukkan Jumlah Rumah Daerah 2 : 6
Masukkan 6 Nomor Rumah Daerah 2 : 6 7 8 9 10 11
Nomor Rumah Terurut (Ganjil - Genap) Daerah 2 : 7 9 11 10 8 6
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> 

```

Deskripsi Program

Program di atas berfungsi untuk memproses data nomor rumah dari beberapa daerah, mengelompokkan nomor rumah menjadi kategori ganjil dan genap, mengurutkan masing-masing kategori sesuai aturan tertentu, dan kemudian menampilkan hasil pengelompokannya dalam format yang rapi.

Algoritma dan Cara Program Berfungsi : Pengguna pertama-tama diminta untuk memasukkan jumlah daerah (variabel t). Untuk setiap daerah, program meminta jumlah rumah (n) dan nomor rumah yang akan diinputkan dalam bentuk array. Selanjutnya, nomor rumah tersebut dipisahkan menjadi dua kelompok: nomor ganjil dan nomor genap. Kelompok nomor ganjil diurutkan dalam urutan menaik menggunakan fungsi sort.Ints, sedangkan kelompok nomor genap diurutkan dalam urutan menurun menggunakan kombinasi fungsi sort.Sort dan sort.Reverse. Setelah proses pengelompokkan dan pengurutan selesai, program menampilkan hasilnya untuk setiap daerah. Nomor ganjil ditampilkan terlebih dahulu dalam urutan menaik, diikuti oleh nomor genap dalam urutan menurun.

2. UNGUIDED 2

Studi Case : Kompetisi pemrograman yang baru saja berlalu diikuti oleh 17 tim dari berbagai perguruan tinggi ternama. Dalam kompetisi tersebut, setiap tim bertomba 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. 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 diurutkan: 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"

// Fungsi untuk mengurutkan array menggunakan metode
seleksi
func urutkanArray(data []int) {
    panjang := len(data)
    for i := 0; i < panjang-1; i++ {
        indeksTerkecil := i
        for j := i + 1; j < panjang; j++ {
            if data[j] < data[indeksTerkecil] {
                indeksTerkecil = j
            }
        }
        // Menukar elemen agar array terurut
        data[i], data[indeksTerkecil] = data[indeksTerkecil],
data[i]
    }
}

// Fungsi untuk menghitung nilai median dari array
func hitungMedian(data []int) int {
    panjang := len(data)
    if panjang%2 == 0 {
        return (data[panjang/2-1] + data[panjang/2]) / 2
    }
    return data[panjang/2]
}

func main() {
    var angka int
    var dataSementara []int
    var kumpulanGrup [][]int

    fmt.Println("Masukkan data (akhiri dengan -5313)")
    for {
        fmt.Scan(&angka)
        if angka == -5313 { // Penghentian input
            break
        }
        if angka == 0 { // Menyimpan salinan data ke dalam
grup baru
            kumpulanGrup = append(kumpulanGrup, append([]int{},
dataSementara...))
        } else {
```

```

        dataSementara = append(dataSementara, angka) //
        Tambahkan angka ke data sementara
    }
}

// Memproses setiap grup data untuk menghitung mediannya
for indeks, grup := range kumpulanGrup {
    urutkanArray(grup) // Urutkan grup data
    nilaiMedian := hitungMedian(grup) // Hitung median
    grup
    fmt.Printf("Median %d : %d\n", indeks+1, nilaiMedian)
}
}

```

Screenshoot Output

```

PROBLEMS 8 OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> go run "d:\KULIAH\SEMESTER 3\
Masukkan data (akhiri dengan -5313)
7 23 11 0 5 19 2 29 3 13 17 0 -5313
Median 1 : 11
Median 2 : 12
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13>

```

Deskripsi Program

Program di atas dirancang untuk menerima data angka secara bertahap, mengelompokkan data ke dalam beberapa grup berdasarkan pemisah angka nol, dan menghitung nilai median dari masing-masing grup.

Algoritma dan Cara Program Berfungsi : Prosesnya dimulai dengan meminta pengguna untuk memasukkan angka secara berulang, dan input dihentikan ketika pengguna memasukkan angka -5313. Setiap kali angka nol dimasukkan, program menyimpan data yang sudah dimasukkan sebelumnya sebagai satu grup terpisah ke dalam variabel kumpulanGrup, kemudian melanjutkan dengan mengumpulkan data baru. Data sementara yang dikumpulkan direset untuk memulai grup baru setelah angka nol. Setelah semua data dimasukkan, program memproses setiap grup yang ada dalam kumpulanGrup. Untuk setiap grup, program menggunakan metode pengurutan seleksi (selection sort) untuk mengurutkan elemen-elemen dari yang terkecil hingga terbesar. Setelah data dalam grup terurut, program menghitung nilai median menggunakan fungsi hitungMedian. Median dihitung dengan membedakan kasus data ganjil (elemen tengah) dan data genap (rata-rata dua elemen tengah). Program kemudian mencetak nilai median dari setiap grup dengan format yang rapi.

3. UNGUIDED 3

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"
)

// Struct untuk menyimpan informasi buku
type Buku struct {
    ID          string
    Judul       string
    Penulis     string
    Penerbit    string
    Eksemplar   int
    Tahun       int
    Rating      int
}

// Struct untuk menyimpan koleksi buku
type DaftarBuku struct {
    Pustaka []Buku
    NPustaka int
}

// Fungsi untuk menambahkan buku ke daftar pustaka
func TambahBuku(pustaka *DaftarBuku) {
    var jumlah int
    fmt.Print("Masukkan Jumlah Buku : ")
    fmt.Scan(&jumlah)

    pustaka.NPustaka = jumlah
    pustaka.Pustaka = make([]Buku, jumlah)

    for i := 0; i < jumlah; i++ {
        i+1)
        fmt.Print("ID : ")
        fmt.Scan(&pustaka.Pustaka[i].ID)
        fmt.Print("Judul : ")
        fmt.Scan(&pustaka.Pustaka[i].Judul)
        fmt.Print("Penulis : ")
        fmt.Scan(&pustaka.Pustaka[i].Penulis)
        fmt.Print("Penerbit : ")
        fmt.Scan(&pustaka.Pustaka[i].Penerbit)
        fmt.Print("Eksemplar : ")
        fmt.Scan(&pustaka.Pustaka[i].Eksemplar)
        fmt.Print("Tahun : ")
        fmt.Scan(&pustaka.Pustaka[i].Tahun)
        fmt.Print("Rating : ")
        fmt.Scan(&pustaka.Pustaka[i].Rating)
    }
}
```

```

}

// Fungsi untuk mencetak semua data buku dalam format tabel
func CetakBuku(pustaka DaftarBuku) {
    fmt.Println("\nDaftar Buku :")
    fmt.Printf("%-5s %-25s %-20s %-20s %-10s %-10s %-5s\n", "ID", "Judul", "Penulis", "Penerbit", "Eksemplar", "Tahun", "Rating")
    for _, buku := range pustaka.Pustaka {
        fmt.Printf("%-5s %-25s %-20s %-20s %-10d %-10d %-5d\n",
                    buku.ID, buku.Judul, buku.Penulis,
                    buku.Penerbit, buku.Eksemplar, buku.Tahun, buku.Rating)
    }
}

// Fungsi untuk mengurutkan buku berdasarkan rating secara descending
func UrutkanBuku(pustaka *DaftarBuku) {
    n := pustaka.NPustaka
    for i := 1; i < n; i++ {
        key := pustaka.Pustaka[i]
        j := i - 1
        for j >= 0 && pustaka.Pustaka[j].Rating < key.Rating {
            pustaka.Pustaka[j+1] = pustaka.Pustaka[j]
            j--
        }
        pustaka.Pustaka[j+1] = key
    }
}

// Fungsi untuk mencetak 5 buku dengan rating tertinggi
func CetakBukuTeratas(pustaka DaftarBuku) {
    fmt.Println("\n5 Buku dengan Rating Tertinggi :")
    fmt.Printf("%-25s %-5s\n", "Judul", "Rating")
    for i := 0; i < 5 && i < pustaka.NPustaka; i++ {
        fmt.Printf("%-25s %-5d\n",
                    pustaka.Pustaka[i].Judul, pustaka.Pustaka[i].Rating)
    }
}

// Fungsi untuk mencari buku berdasarkan rating menggunakan pencarian biner
func CariBuku(pustaka DaftarBuku) {
    var rating int
    fmt.Print("\nMasukkan rating yang ingin dicari : ")
    fmt.Scan(&rating)

    kiri, kanan := 0, pustaka.NPustaka-1
    ditemukan := false
    for kiri <= kanan {

```

```

        tengah := (kiri + kanan) / 2
        if pustaka.Pustaka[tengah].Rating == rating {
            ditemukan = true
            buku := pustaka.Pustaka[tengah]
            fmt.Printf("\nBuku dengan Rating %d :\n",
rating)
                fmt.Printf("%-5s %-25s %-20s %-20s %-10s %-
10s %-5s\n", "ID", "Judul", "Penulis", "Penerbit",
"Eksemplar", "Tahun", "Rating")
                fmt.Printf("%-5s %-25s %-20s %-20s %-10d %-
10d %-5d\n",
                    buku.ID, buku.Judul, buku.Penulis,
buku.Penerbit, buku.Eksemplar, buku.Tahun, buku.Rating)
                break
            } else if pustaka.Pustaka[tengah].Rating < rating
{
                kanan = tengah - 1
            } else {
                kiri = tengah + 1
            }
        }
        if !ditemukan {
            fmt.Println("\nBuku dengan rating tersebut tidak
ditemukan.")
        }
    }

func main() {
    var pustaka DaftarBuku

    // Tambah data buku
    TambahBuku(&pustaka)

    // Urutkan buku berdasarkan rating
    UrutkanBuku(&pustaka)

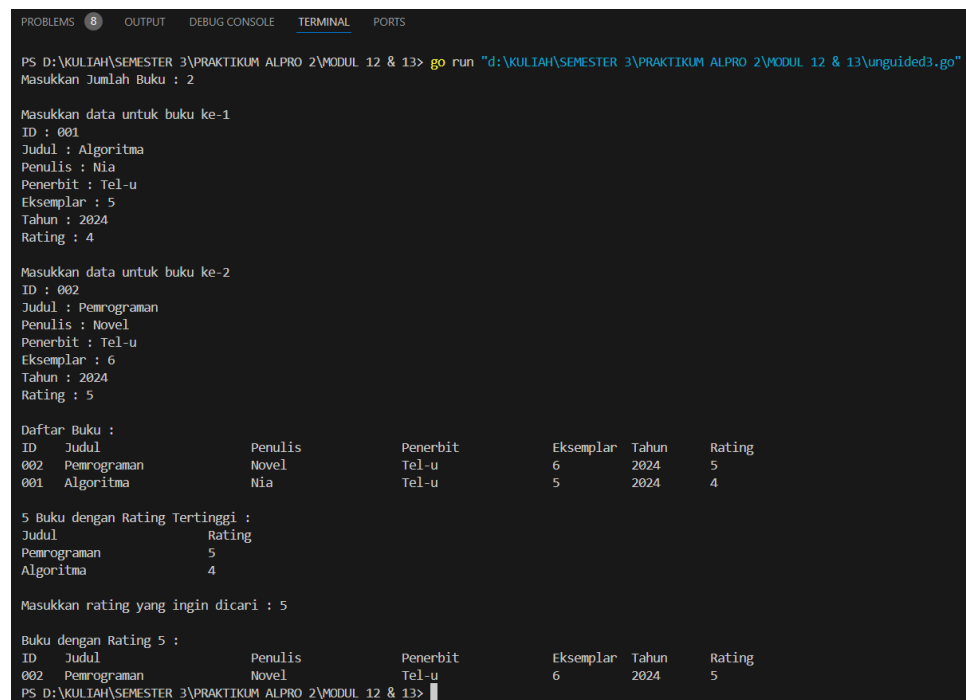
    // Cetak semua data buku
    CetakBuku(pustaka)

    // Cetak 5 buku dengan rating tertinggi
    CetakBukuTeratas(pustaka)

    // Cari buku berdasarkan rating
    CariBuku(pustaka)
}

```

Screenshoot Output



```
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> go run "d:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13\unguided3.go"
Masukkan Jumlah Buku : 2

Masukkan data untuk buku ke-1
ID : 001
Judul : Algoritma
Penulis : Nia
Penerbit : Tel-u
Eksemplar : 5
Tahun : 2024
Rating : 4

Masukkan data untuk buku ke-2
ID : 002
Judul : Pemrograman
Penulis : Novel
Penerbit : Tel-u
Eksemplar : 6
Tahun : 2024
Rating : 5

Daftar Buku :
ID      Judul      Penulis      Penerbit      Eksemplar      Tahun      Rating
002     Pemrograman  Novel       Tel-u         6              2024       5
001     Algoritma   Nia         Tel-u         5              2024       4

5 Buku dengan Rating Tertinggi :
Judul      Rating
Pemrograman 5
Algoritma  4

Masukkan rating yang ingin dicari : 5

Buku dengan Rating 5 :
ID      Judul      Penulis      Penerbit      Eksemplar      Tahun      Rating
002     Pemrograman  Novel       Tel-u         6              2024       5
PS D:\KULIAH\SEMESTER 3\PRAKTIKUM ALPRO 2\MODUL 12 & 13> |
```

Deskripsi Program

Program di atas adalah sistem manajemen data buku yang dirancang untuk mencatat, mengurutkan, mencari, dan menampilkan informasi buku berdasarkan rating. Program ini menggunakan struktur data struct untuk merepresentasikan informasi buku dan koleksi pustaka. Informasi yang disimpan untuk setiap buku mencakup ID, judul, penulis, penerbit, jumlah eksemplar, tahun terbit, dan rating.

Algoritma dan Cara Program Berfungsi : Program dimulai dengan meminta pengguna memasukkan jumlah buku, kemudian data setiap buku dimasukkan melalui fungsi `TambahBuku`. Data yang dimasukkan disimpan dalam variabel `DaftarBuku`, yang berisi array dari objek buku. Setelah data dikumpulkan, fungsi `UrutkanBuku` digunakan untuk mengurutkan daftar buku berdasarkan rating secara menurun menggunakan algoritma *insertion sort*. Hasil pengurutan membantu memastikan buku dengan rating tertinggi berada di awal daftar. Setelah itu, program menampilkan seluruh data buku dalam format tabel melalui fungsi `CetakBuku`. Program juga mencetak lima buku dengan rating tertinggi menggunakan fungsi `CetakBukuTeratas`. Fungsi ini memanfaatkan hasil pengurutan sehingga buku-buku dengan rating tertinggi langsung diambil dari awal daftar. Fitur pencarian buku berdasarkan rating juga disediakan melalui fungsi `CariBuku`. Fungsi ini menggunakan algoritma pencarian biner (*binary search*) untuk menemukan buku dengan rating tertentu. Algoritma ini bekerja dengan membagi daftar

menjadi dua bagian pada setiap langkah, sehingga pencarian lebih efisien. Jika buku dengan

IV. KESIMPULAN

Kesimpulan dari mempelajari materi *selection sort* dan *insertion sort* adalah bahwa keduanya merupakan algoritma pengurutan sederhana yang efektif untuk menyusun elemen dalam daftar secara terurut. *Selection sort* bekerja dengan cara memilih elemen terkecil dari bagian yang belum terurut dan menukarnya dengan elemen pada posisi yang sesuai. Keunggulan utama dari algoritma ini adalah kemudahan implementasi dan kecocokannya untuk dataset kecil, meskipun algoritma ini kurang efisien pada dataset besar karena tetap melakukan iterasi yang sama, meskipun data sudah terurut sebagian. Sementara itu, *insertion sort* menyisipkan elemen pada posisi yang tepat dalam bagian yang sudah terurut. Keunggulan dari *insertion sort* adalah kinerjanya yang efisien pada data yang hampir terurut dan kemampuannya menangani dataset kecil dengan baik. Namun, seperti halnya *selection sort*, kinerjanya menurun pada dataset yang besar karena membutuhkan banyak pergeseran elemen. Pemilihan antara kedua algoritma ini tergantung pada karakteristik dataset dan kebutuhan aplikasi; untuk data kecil atau hampir terurut, *insertion sort* lebih efisien, sementara *selection sort* dapat digunakan untuk situasi yang lebih sederhana atau ketika stabilitas algoritma bukan prioritas. Memahami kedua algoritma ini memberikan dasar yang kuat untuk mempelajari teknik pengurutan yang lebih kompleks.

V. REFERENSI

- [1] Modul XII & XIII Praktikum Algoritma dan Pemrograman 2
- [2] <https://www.programiz.com/dsa/selection-sort>
- [3] <https://www.programiz.com/dsa/insertion-sort>