LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL XII & XIII

PENGURUTAN DATA

Oleh:

Muhammad Rifki Fadhilah

2311102032

IF 11 02

S1 TEKNIK INFORMATIKA

INSTITUT TEKNOLOGI TELKOM PURWOKERTO

2024/2025

I. DASAR TEORI

Selection Sort

Selection sort adalah algoritma pengurutan sederhana yang bekerja dengan cara menemukan elemen terkecil (atau terbesar, tergantung kebutuhan) dalam array dan menukarnya dengan elemen pada posisi awal yang belum diurutkan. Proses ini diulang hingga seluruh elemen terurut. Berikut adalah implementasinya:

```
func selectionSort(arr []int) {
  n := len(arr)
  for i := 0; i < n-1; i++ {
      minIdx := i
      for j := i + 1; j < n; j++ {
          if arr[j] < arr[minIdx] { // Cari elemen terkecil
              minIdx = j
          }
      }
      arr[i], arr[minIdx] = arr[minIdx], arr[i] // Tukar
elemen
  }
}
```

Insertion Sort

Insertion sort adalah algoritma pengurutan yang bekerja dengan cara membangun array yang sudah terurut secara bertahap. Elemen diproses satu per satu dan disisipkan ke posisi yang sesuai dalam bagian array yang sudah diurutkan.Berikut adalah implementasinya:

```
func insertionSort(arr []int) {
  n := len(arr)
  for i := 1; i < n; i++ {
      key := arr[i]
      j := i - 1
      for j >= 0 && arr[j] > key { // Geser elemen lebih besar
          arr[j+1] = arr[j]
          j--
      }
      arr[j+1] = key // Tempatkan elemen pada posisi yang
sesuai
  }
}
```

II. GUIDED

1. SOURCE CODE

```go
package main

import "fmt"

// Fungsi untuk mengurutkan array menggunakan selection sort
func selectionSort(arr []int) {
  n := len(arr)
  for i := 0; i < n-1; i++ {
    maxIdx := i
    for j := i + 1; j < n; j++ {
      if arr[j] > arr[maxIdx] { // Cari elemen terbesar
        maxIdx = j
      }
    }
    arr[i], arr[maxIdx] = arr[maxIdx], arr[i] // Tukar elemen
  }
}

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

  if n <= 0 || n >= 1000 {
    fmt.Println("n harus lebih besar dari 0 dan kurang dari
1000.")
    return
  }

  for i := 0; i < n; i++ {
    var m int
    fmt.Printf("Masukkan jumlah rumah kerabat untuk daerah ke-%d:
", i+1)
    fmt.Scan(&m)

    if m <= 0 || m >= 1000000 {
      fmt.Println("m harus lebih besar dari 0 dan kurang dari
1000000.")
      return
    }
```

```
    // Masukkan nomor rumah
    houses := make([]int, m)
    fmt.Printf("Masukkan nomor rumah kerabat untuk daerah ke-%d:
", i+1)
    for j := 0; j < m; j++ {
      fmt.Scan(&houses[j])
    }

    // Urutkan dengan selection sort
    selectionSort(houses)

    // Cetak hasil
    fmt.Printf("Hasil urutan rumah untuk daerah ke-%d: ", i+1)
    for _, house := range houses {
      fmt.Printf("%d ", house)
    }
    fmt.Println()
  }
}
```

OUTPUT

```
PS D:\Project VS Code\golang\Alpro\Modul 12> go run "d:\Project VS Code\golang\Alpro\Modul 12\guided\1\main.go"
Masukkan jumlah daerah (n): 3
Masukkan jumlah rumah kerabat untuk daerah ke-1: 5 2 1 7 9 13
Masukkan nomor rumah kerabat untuk daerah ke-1: Hasil urutan rumah untuk daerah ke-1: 13 9 7 2 1
Masukkan jumlah rumah kerabat untuk daerah ke-2: 6 189 15 27 39 75 133
Masukkan nomor rumah kerabat untuk daerah ke-2: Hasil urutan rumah untuk daerah ke-2: 189 133 75 39 27 15
Masukkan jumlah rumah kerabat untuk daerah ke-3: 3 4 9 1
Masukkan nomor rumah kerabat untuk daerah ke-3: Hasil urutan rumah untuk daerah ke-3: 9 4 1
PS D:\Project VS Code\golang\Alpro\Modul 12>
```

DESKRIPSI PROGRAM

Program ini ditulis dalam bahasa Go untuk mengelola data nomor rumah kerabat di beberapa daerah. Program ini menerima jumlah daerah yang akan diproses, diikuti dengan data nomor rumah kerabat di setiap daerah, dan mengurutkan nomor-nomor rumah tersebut dalam urutan menurun menggunakan algoritma **selection sort**.

Setelah dimulai, program meminta pengguna untuk memasukkan jumlah daerah yang akan diproses, yaitu nnn. Nilai nnn harus memenuhi syarat antara 1 dan 999, karena nilai di luar rentang tersebut akan dihentikan dengan pesan peringatan. Untuk setiap daerah, pengguna kemudian diminta untuk memasukkan jumlah rumah kerabat mmm, yang juga harus berada di rentang antara 1

dan 999,999. Jika mmm tidak memenuhi syarat, program akan menghentikan proses dan memberi tahu pengguna tentang batasan tersebut.

Untuk setiap daerah, program meminta pengguna memasukkan nomor rumah kerabat dalam bentuk array. Nomor rumah ini kemudian diurutkan dalam urutan menurun menggunakan algoritma **selection sort**. Algoritma ini bekerja dengan mencari elemen terbesar dari array yang belum diurutkan dan menukarnya ke posisi yang sesuai di bagian awal array. Proses ini diulang untuk setiap elemen dalam array hingga seluruhnya terurut.

Setelah proses pengurutan selesai untuk sebuah daerah, program mencetak nomor-nomor rumah yang telah diurutkan ke layar. Proses ini diulang untuk setiap daerah hingga semua daerah selesai diproses.

Dengan pendekatan ini, program memastikan bahwa data rumah kerabat di setiap daerah dapat diurutkan dengan mudah dan ditampilkan dalam format yang terorganisasi. Validasi input yang ketat juga menjamin bahwa data yang dimasukkan sesuai dengan batasan yang telah ditentukan.

2. SOURCE CODE

```go
package main

import (
  "fmt"
  "math"
)

// Fungsi insertion sort untuk mengurutkan array
func insertionSort(arr []int) {
  n := len(arr)
  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 data berjarak tetap
func isDataConsistentlySpaced(arr []int) (bool, int) {
  if len(arr) < 2 {
    return true, 0 // Array dengan kurang dari 2 elemen dianggap
berjarak tetap
  }

  // Hitung selisih awal
  diff := int(math.Abs(float64(arr[1] - arr[0])))

  for i := 1; i < len(arr)-1; i++ {
    currentDiff := int(math.Abs(float64(arr[i+1] - arr[i])))
    if currentDiff != diff {
      return false, 0 // Jika ada selisih yang berbeda, tidak
berjarak tetap
    }
  }

  return true, diff
}
```

```go
func main() {
  var data []int
  var input int

  fmt.Println("Masukkan data (akhiri dengan bilangan negatif):")
  for {
    fmt.Scan(&input)
    if input < 0 {
      break
    }
    data = append(data, input)
  }

  // Urutkan data menggunakan insertion sort
  insertionSort(data)

  // Periksa apakah data berjarak tetap
  isConsistent, diff := isDataConsistentlySpaced(data)

  // Cetak hasil
  fmt.Println("Hasil pengurutan:", data)
  if isConsistent {
    fmt.Printf("Data berjarak %d\n", diff)
  } else {
    fmt.Println("Data berjarak tidak tetap")
  }
}
```

OUTPUT

```
PS D:\Project VS Code\golang\Alpro\Modul 12> go run
Masukkan data (akhiri dengan bilangan negatif):
4 40 14 8 26 1 38 2 32 -31
Hasil pengurutan: [1 2 4 8 14 26 32 38 40]
Data berjarak tidak tetap
```

DESKRIPSI PROGRAM

Program ini ditulis dalam bahasa Go untuk menerima serangkaian data, mengurutkannya, dan memeriksa apakah data tersebut memiliki jarak tetap antara elemen-elemen setelah diurutkan. Program ini memanfaatkan algoritma **insertion sort** untuk mengurutkan data dan logika matematis sederhana untuk menghitung perbedaan antara elemen-elemen bertetangga.
Saat dijalankan, program meminta pengguna untuk memasukkan data berupa bilangan bulat secara berurutan. Pengguna dapat

mengakhiri proses input dengan memasukkan bilangan negatif. Semua bilangan positif yang dimasukkan oleh pengguna disimpan ke dalam sebuah array.

Setelah data terkumpul, program mengurutkannya menggunakan **insertion sort**. Algoritma ini bekerja dengan membandingkan elemen-elemen array secara berurutan dan menyisipkannya ke posisi yang sesuai dalam bagian array yang sudah diurutkan. Dengan cara ini, data akan diatur dalam urutan menaik.

Setelah data diurutkan, program memeriksa apakah jarak antar elemen (selisih absolut antara elemen bertetangga) bersifat tetap di seluruh array. Jika array memiliki kurang dari dua elemen, data dianggap otomatis berjarak tetap karena tidak ada elemen lain untuk diperbandingkan. Untuk array dengan lebih dari satu elemen, program menghitung selisih awal antara dua elemen pertama dan membandingkan selisih tersebut dengan selisih antara elemen-elemen lainnya. Jika semua selisih sama, data dianggap berjarak tetap, dan jarak tetap tersebut dicetak. Namun, jika ditemukan perbedaan dalam selisih, program menyimpulkan bahwa data tidak memiliki jarak tetap.

Hasil dari program mencakup dua hal utama: data yang telah diurutkan dalam urutan menaik dan informasi apakah data tersebut memiliki jarak tetap atau tidak. Jika jarak tetap terdeteksi, jarak tersebut juga ditampilkan untuk memberikan informasi tambahan kepada pengguna.

III. UNGUIDED

1. SOURCE CODE

```
package main

import "fmt"

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

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

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

  if n <= 0 || n >= 1000 {
    fmt.Println("n harus lebih besar dari 0 dan kurang dari
1000.")
    return
  }

  for i := 0; i < n; i++ {
    var m int
    fmt.Printf("Masukkan jumlah rumah kerabat untuk daerah ke-%d:
", i+1)
    fmt.Scan(&m)
```

```
    if m <= 0 || m >= 1000000 {
      fmt.Println("m harus lebih besar dari 0 dan kurang dari
1000000.")
      return
    }

    houses := make([]int, m)
    fmt.Printf("Masukkan nomor rumah kerabat untuk daerah ke-%d:
", i+1)
    for j := 0; j < m; j++ {
      fmt.Scan(&houses[j])
    }

    var oddNumbers, evenNumbers []int
    for _, house := range houses {
      if house%2 == 1 {
        oddNumbers = append(oddNumbers, house)
      } else {
        evenNumbers = append(evenNumbers, house)
      }
    }

    selectionSortAsc(oddNumbers)

    selectionSortDesc(evenNumbers)

    fmt.Printf("Hasil urutan rumah untuk daerah ke-%d: ", i+1)

    for _, house := range oddNumbers {
      fmt.Printf("%d ", house)
    }

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

OUTPUT

```
PS D:\Project VS Code\golang\Alpro\Modul 12> go run "d:\Project VS Code\golang\Alpro\Modul 12\unguided\1\main.go"
Masukkan jumlah daerah (n): 3
Masukkan jumlah rumah kerabat untuk daerah ke-1: 5 2 1 7 9 13
Masukkan nomor rumah kerabat untuk daerah ke-1: Hasil urutan rumah untuk daerah ke-1: 1 7 9 13 2
Masukkan jumlah rumah kerabat untuk daerah ke-2: 6 189 15 27 39 75 133
Masukkan nomor rumah kerabat untuk daerah ke-2: Hasil urutan rumah untuk daerah ke-2: 15 27 39 75 133 189
Masukkan jumlah rumah kerabat untuk daerah ke-3: 3 4 9 1
Masukkan nomor rumah kerabat untuk daerah ke-3: Hasil urutan rumah untuk daerah ke-3: 1 9 4
```

DESKRIPSI PROGRAM

Program ini bertujuan untuk mengelompokkan dan mengurutkan nomor rumah berdasarkan kategori ganjil dan genap untuk beberapa daerah. Proses pengolahan data dilakukan melalui input manual dari pengguna, di mana pengguna memasukkan jumlah daerah, jumlah rumah kerabat di setiap daerah, dan nomor rumah kerabat tersebut.

Setelah jumlah daerah dimasukkan, program meminta jumlah rumah kerabat untuk masing-masing daerah. Jika jumlah yang dimasukkan tidak valid, program akan memberikan peringatan dan berhenti. Nomor rumah kerabat untuk setiap daerah juga dimasukkan oleh pengguna, dengan setiap nomor disimpan dalam sebuah slice. Program kemudian memisahkan nomor-nomor rumah menjadi dua kelompok: ganjil dan genap. Angka ganjil disimpan dalam slice khusus, sementara angka genap disimpan dalam slice lainnya.

Selanjutnya, program mengurutkan kedua kelompok ini dengan menggunakan algoritma selection sort. Nomor rumah ganjil diurutkan dalam urutan menaik, sedangkan nomor rumah genap diurutkan dalam urutan menurun. Hasil dari pengurutan ini adalah dua kelompok yang terorganisasi dengan baik, masing-masing berdasarkan kategori ganjil dan genap.

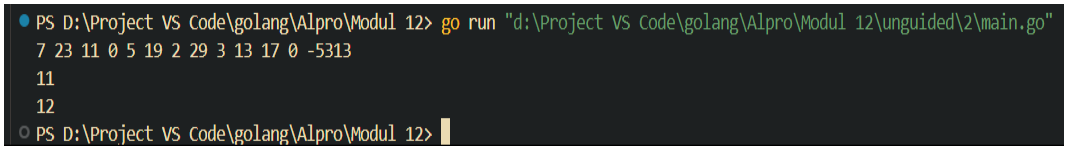
Setelah pengurutan selesai, program mencetak hasil pengelompokan dan pengurutan untuk setiap daerah. Nomor rumah ganjil ditampilkan terlebih dahulu dalam urutan menaik, diikuti oleh nomor rumah genap dalam urutan menurun. Proses ini dilakukan secara terpisah untuk setiap daerah yang dimasukkan pengguna, sehingga setiap daerah memiliki hasil pengelompokan dan pengurutan nomor rumah yang unik.

Program memastikan semua masukan sesuai dengan batasan yang telah ditentukan, seperti jumlah daerah tidak boleh lebih dari 1000 dan jumlah rumah per daerah tidak boleh lebih dari 1.000.000. Validasi ini membantu menjaga keakuratan dan kestabilan proses pengolahan data. Dengan alur ini, program memberikan hasil

pengelompokan dan pengurutan nomor rumah yang terstruktur dengan baik untuk setiap daerah.

2. SOURCE CODE

```go
package main

import "fmt"

func hitungMedian(arr []int) int {
    n := len(arr)
    if n%2 == 1 {
        return arr[n/2]
    }
    return (arr[n/2-1] + arr[n/2]) / 2
}

func insertionSort(arr []int) {
    for i := 1; i < len(arr); i++ {
        key := arr[i]
        j := i - 1
        for j >= 0 && arr[j] > key {
            arr[j+1] = arr[j]
            j = j - 1
        }
        arr[j+1] = key
    }
}

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

    for {
        fmt.Scan(&input)
        if input == -5313 {
            break
        }

        if input == 0 {
            insertionSort(data)
            median := hitungMedian(data)
            fmt.Println(median)
        } else {
            data = append(data, input)
        }
    }
}
```



DESKRIPSI PROGRAM

Program ini dirancang untuk membaca serangkaian angka, menghitung median dari angka-angka tersebut setiap kali pengguna memasukkan angka nol, dan berhenti jika pengguna memasukkan angka -5313. Median dihitung sebagai nilai tengah dalam sekumpulan data yang telah diurutkan. Jika jumlah data ganjil, median adalah elemen di posisi tengah. Namun, jika jumlah data genap, median dihitung sebagai rata-rata dari dua elemen tengah.

Proses dimulai dengan membaca input dari pengguna secara berulang. Angka-angka yang dimasukkan selain nol dan -5313 akan disimpan dalam sebuah slice bernama data. Jika pengguna memasukkan angka nol, program akan mengurutkan slice tersebut menggunakan algoritma insertion sort. Algoritma ini bekerja dengan cara menyisipkan setiap elemen pada posisi yang tepat untuk menjaga data tetap terurut.

Setelah data diurutkan, program menghitung median dari data tersebut. Jika jumlah elemen dalam slice ganjil, elemen di posisi tengah diambil sebagai median. Jika jumlah elemen genap, median dihitung dengan menjumlahkan dua elemen tengah lalu membagi hasilnya dengan dua. Median yang telah dihitung kemudian ditampilkan kepada pengguna.

Proses ini akan terus berlangsung hingga pengguna memasukkan angka -5313, yang berfungsi sebagai tanda untuk mengakhiri program. Program ini memastikan bahwa angka nol hanya digunakan untuk memicu penghitungan dan tidak disimpan sebagai bagian dari data. Dengan cara ini, program memberikan median yang akurat berdasarkan data yang dimasukkan sebelum angka nol.

3. SOURCE CODE

```go
package main

import (
  "fmt"
)

const nMax = 7919
type Buku struct{
  id, judul, penulis, penerbit string
  eksemplar, tahun, rating int
}

type DaftarBuku = [nMax]Buku

func DaftarkanBuku(pustaka *DaftarBuku ,n int){
  for i := 0; i < n; i++ {
    fmt.Printf("Buku %v: \n",i+1)
    fmt.Print("Masukkan id buku: ")
    fmt.Scan(&pustaka[i].id)
    fmt.Print("Masukkan judul buku: ")
    fmt.Scan(&pustaka[i].judul)
    fmt.Print("Masukkan penulis buku: ")
    fmt.Scan(&pustaka[i].penulis)
    fmt.Print("Masukkan penerbit buku: ")
    fmt.Scan(&pustaka[i].penerbit)
    fmt.Print("Masukkan eksemplar buku: ")
    fmt.Scan(&pustaka[i].eksemplar)
    fmt.Print("Masukkan tahun terbit: ")
    fmt.Scan(&pustaka[i].tahun)
    fmt.Print("Masukkan rating buku: ")
    fmt.Scan(&pustaka[i].rating)
    fmt.Println()
  }
}

func CetakTerfavorit(pustaka DaftarBuku ,n int){
  max := pustaka[0]
  for i := 1; i < n; i++ {
    if pustaka[i].rating > max.rating{
      max = pustaka[i]
    }
  }
```

```
  fmt.Println("Buku Favorit: ")
  fmt.Printf("Judul buku: %v\n", max.judul)
  fmt.Printf("Penulis buku: %v\n", max.penulis)
  fmt.Printf("Penerbit buku: %v\n", max.penerbit)
  fmt.Println()
}

func UrutBuku(pustaka DaftarBuku ,n int){
  for i := 1; i < n; i++ {
    key := (pustaka)[i]
    j := i - 1

    for j >= 0 && (pustaka)[j].rating > key.rating{
      pustaka[j+1] = (pustaka)[j]
      j--
    }
    (pustaka)[j+1] = key
  }
}

func Cetak5Terbaru(pustaka DaftarBuku ,n int){
  for i := 1; i < n; i++ {
    key := (pustaka)[i]
    j := i - 1

    for j >= 0 && (pustaka)[j].rating < key.rating{
      pustaka[j+1] = (pustaka)[j]
      j--
    }
    (pustaka)[j+1] = key
  }
  fmt.Println("5 Judul Buku dengan Rating Tertinggi: ")
  limit := 5
  if n < limit {
    limit = n
  }
  for i := 0; i < limit; i++ {
    fmt.Printf("Buku %v: \n",i+1)
    fmt.Printf("Judul buku: %v\n", pustaka[i].judul)
    fmt.Println()
  }
}

func CariBuku(pustaka DaftarBuku, n, r int) {
  fmt.Print("Masukkan rating untuk mencari buku: ")
  fmt.Scan(&r)
```

```
  low, high := 0, n-1
  found := false
  for low <= high {
    mid := (low + high) / 2
    if pustaka[mid].rating == r {
      fmt.Println("Buku dengan rating", r, ":")

      i := mid
      for i >= 0 && pustaka[i].rating == r {
        fmt.Printf("Buku %v: \n", i+1)
        fmt.Printf("Judul buku: %v\n", pustaka[i].judul)
        fmt.Printf("Penulis buku: %v\n", pustaka[i].penulis)
        fmt.Printf("Penerbit buku: %v\n", pustaka[i].penerbit)
        fmt.Printf("Tahun terbit: %v\n", pustaka[i].tahun)
        fmt.Println()
        i--
      }

      i = mid + 1
      for i < n && pustaka[i].rating == r {
        fmt.Printf("Buku %v: \n", i+1)
        fmt.Printf("Judul buku: %v\n", pustaka[i].judul)
        fmt.Printf("Penulis buku: %v\n", pustaka[i].penulis)
        fmt.Printf("Penerbit buku: %v\n", pustaka[i].penerbit)
        fmt.Printf("Tahun terbit: %v\n", pustaka[i].tahun)
        fmt.Println()
        i++
      }
      found = true
      break
    } else if pustaka[mid].rating < r {
      low = mid + 1
    } else {
      high = mid - 1
    }
  }
  if !found {
    fmt.Println("Tidak ada buku dengan rating seperti itu")
  }
}
```

```
func main(){
  var pustaka DaftarBuku
  var Npustaka,rating int

  fmt.Print("Masukkan jumlah buku: ")
  fmt.Scan(&Npustaka)
```

```
  DaftarkanBuku(&pustaka, Npustaka)
  CetakTerfavorit(pustaka, Npustaka)
  UrutBuku(pustaka,Npustaka)
  Cetak5Terbaru(pustaka,Npustaka)
  CariBuku(pustaka,Npustaka,rating)
}
```

OUTPUT

```
Masukkan jumlah buku: 5
Buku 1:
Masukkan id buku: 1
Masukkan judul buku: Si_Kancil
Masukkan penulis buku: Andi
Masukkan penerbit buku: Pustaka_Buku
Masukkan eksemplar buku: 6
Masukkan tahun terbit: 2023
Masukkan rating buku: 6

Buku 2:
Masukkan id buku: 2
Masukkan judul buku: Belajar_Bahasa_Cina
Masukkan penulis buku: Luna
Masukkan penerbit buku: Pustaka_Buku
Masukkan eksemplar buku: 1
Masukkan tahun terbit: 2023
Masukkan rating buku: 4

Buku 3:
Masukkan id buku: 3
Masukkan judul buku: Mengapa_menjadi_laki-laki_itu_berat?
Masukkan penulis buku: Fadhil
Masukkan penerbit buku: Pustaka_Buku
Masukkan eksemplar buku: 2000
Masukkan tahun terbit: 2023
Masukkan rating buku: 8

Buku 4:
Masukkan id buku: 4
Masukkan judul buku: Belajar_Bahasa_Inggris_Untuk_Pemula
Masukkan penulis buku: Luna
Masukkan penerbit buku: Pustaka_Buku
Masukkan eksemplar buku: 1
Masukkan tahun terbit: 2024
Masukkan rating buku: 6

Buku 5:
Masukkan id buku: 5
Masukkan judul buku: Pemrograman_Go
Masukkan penulis buku: Tono
Masukkan penerbit buku: Pustaka_Buku
Masukkan eksemplar buku: 3
Masukkan tahun terbit: 2023
Masukkan rating buku: 5
```

```
Buku Favorit:
Judul buku: Mengapa_menjadi_laki-laki_itu_berat?
Penulis buku: Fadhil
Penerbit buku: Pustaka_Buku

5 Judul Buku dengan Rating Tertinggi:
Buku 1:
Judul buku: Mengapa_menjadi_laki-laki_itu_berat?

Buku 2:
Judul buku: Si_Kancil

Buku 3:
Judul buku: Belajar_Bahasa_Inggris_Untuk_Pemula

Buku 4:
Judul buku: Pemrograman_Go

Buku 5:
Judul buku: Belajar_Bahasa_Cina

Masukkan rating untuk mencari buku: 6
Buku dengan rating 6 :
Buku 1:
Judul buku: Si_Kancil
Penulis buku: Andi
Penerbit buku: Pustaka_Buku
Tahun terbit: 2023
```

DESKRIPSI PROGRAM

Program ini dirancang untuk mengelola data buku dalam sebuah pustaka menggunakan bahasa pemrograman Go. Program ini memungkinkan pengguna untuk mendaftarkan buku, mencetak buku favorit berdasarkan rating, mengurutkan buku berdasarkan rating, dan mencari buku dengan rating tertentu.

Di dalam program ini, terdapat beberapa komponen utama. Pertama, ada tipe data Buku yang memiliki beberapa atribut seperti id, judul, penulis, penerbit, eksemplar, tahun, dan rating. Tipe data ini mewakili informasi yang berkaitan dengan buku. Selanjutnya, ada tipe data DaftarBuku, yang merupakan sebuah array yang menampung hingga 7919 buku.

Pada fungsi DaftarkanBuku, program meminta input dari pengguna untuk mengisi informasi mengenai buku-buku yang akan didaftarkan. Pengguna diminta untuk mengisi id buku, judul buku, penulis, penerbit, jumlah eksemplar, tahun terbit, dan rating untuk setiap buku. Informasi ini kemudian disimpan dalam array DaftarBuku.

Setelah buku-buku didaftarkan, program akan mencari buku favorit berdasarkan rating tertinggi menggunakan fungsi CetakTerfavorit. Fungsi ini akan mencari buku dengan rating tertinggi dan mencetak informasi terkait buku tersebut. Fungsi ini menggunakan algoritma pencarian sederhana untuk menemukan buku dengan rating tertinggi.

Selanjutnya, program mengurutkan buku-buku berdasarkan rating menggunakan fungsi UrutBuku. Fungsi ini menggunakan algoritma penyortiran jenis insertion sort, di mana buku-buku akan diurutkan berdasarkan rating secara ascending. Buku dengan rating lebih tinggi akan diposisikan setelah buku dengan rating lebih rendah.

Program juga memiliki fungsi Cetak5Terbaru, yang mencetak lima buku dengan rating tertinggi. Jika jumlah buku yang dimasukkan kurang dari lima, maka program hanya akan mencetak sebanyak jumlah buku yang ada. Buku-buku ini diurutkan terlebih dahulu berdasarkan rating, dan lima buku pertama yang memiliki rating tertinggi akan ditampilkan.

Terakhir, ada fungsi CariBuku, yang memungkinkan pengguna untuk mencari buku berdasarkan rating tertentu. Pengguna diminta untuk memasukkan rating, dan program akan menggunakan algoritma pencarian biner untuk menemukan buku dengan rating tersebut. Jika buku dengan rating yang diminta ditemukan, maka program akan mencetak semua buku dengan rating yang sama, baik sebelum maupun sesudahnya dalam array.