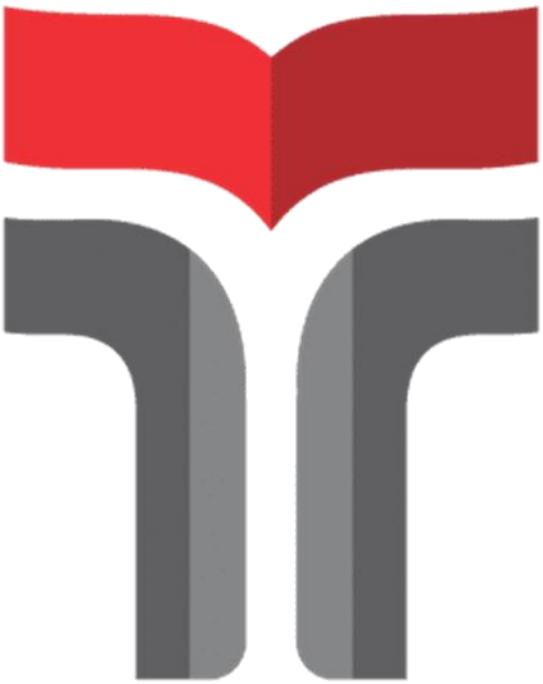
LAPORAN PRAKTIKUM
ALGORITMA PEMROGRAMAN 2

MODUL 11&12

PENGURUTAN DATA (SORTING)



Oleh:

TSAQIF KANZ AHMAD
2311102075
IF-11-02

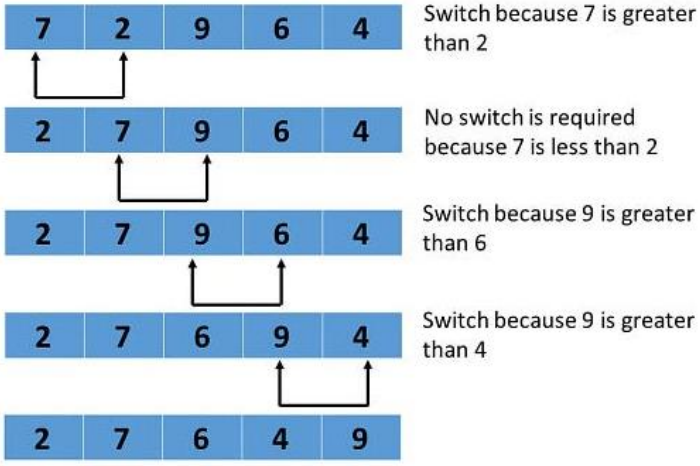
S1 TEKNIK INFORMATIKA
INSTITUT TEKNOLOGI TELKOM
PURWOKERTO
2024

I. DASAR TEORI

Pengurutan (sorting) adalah proses mengatur data dalam urutan tertentu, baik secara *ascending* (menaik) maupun *descending* (menurun). Dalam konteks pemrograman, pengurutan sering digunakan untuk meningkatkan efisiensi dalam pencarian, pengelolaan data, atau analisis. Dalam struktur data, terdapat berbagai metode pengurutan yang digunakan untuk menyusun elemen dalam urutan tertentu. Beberapa algoritma pengurutan yang umum digunakan meliputi :

1. Bubble Sort

Bubble Sort adalah algoritma pengurutan sederhana yang berulang kali menelusuri daftar, membandingkan elemen yang berdekatan, dan menukarnya jika urutannya salah. Proses ini diulang hingga seluruh daftar terurut. Algoritma ini mendapatkan namanya karena elemen yang lebih kecil "menggelembung" ke bagian atas daftar pada setiap iterasi.



Gambar algoritma pada bubble sort

Berikut implementasi pada Bubble sort:

```
func  bubble (array [] int , ukuran int ) {
  untuk i := 0 ; i < ukuran -1 ; i++ {
   untuk j := i + 1 ; j < ukuran; j++ {
   fmt.Println(array[i])
   fmt.Println(array)
    jika array[i] > array[j] {
    array[i], array[j] = array[j], array[i]
   }
  }
}
fmt.Println( "Array yang diurutkan gelembung adalah: " )
fmt.Println(array)
}
```

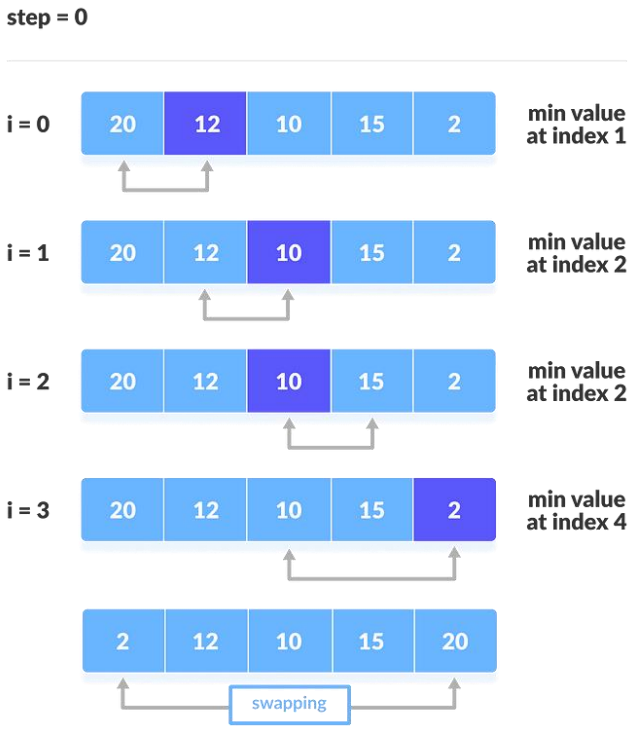
Pada Fungsi ini **bubble** mengambil dua parameter: **array**(array bilangan bulat) dan **size**(jumlah elemen dalam array). Loop luar (**for i := 0; i < size-1; i++**) mengiterasi array dari elemen pertama ke elemen kedua hingga terakhir. Loop ini menunjukkan jumlah lintasan yang diperlukan untuk mengurutkan

array secara menyeluruh. Perulangan dalam (**for j := i + 1; j < size; j++**) mengulangi bagian array yang belum diurutkan, dimulai dari elemen setelah **i** indeks saat ini. Di dalam loop bagian dalam, fungsi tersebut membandingkan **array[i]** dengan **array[j]**. Jika **array[i]** lebih besar dari **array[j]**, berarti urutannya salah, dan pertukaran dilakukan menggunakan fitur penugasan ganda Go (**array[i], array[j] = array[j], array[i]**). Selama setiap iterasi loop dalam, kode menampilkan nilai saat ini dari **array[i]** dan seluruh array menggunakan pernyataan **fmt.Println(array[i])** dan **fmt.Println(array)**. Ini dapat membantu untuk memvisualisasikan perubahan yang terjadi selama setiap lintasan algoritma.

Setelah proses penyortiran selesai, fungsi tersebut mencetak array yang diurutkan menggunakan **fmt.Println(array)**. Untuk menggunakan **bubble** fungsi ini, Anda perlu mengimpor **fmt** paket ke dalam program Go Anda. Selain itu, Anda perlu memanggil **bubble** fungsi tersebut dengan meneruskan array dan ukurannya sebagai argumen. Bubble Sort memiliki kompleksitas waktu $O(n^2)$ dalam kasus terburuk dan rata-rata, di mana n adalah jumlah elemen dalam daftar. Ini tidak dianggap sebagai algoritma yang efisien untuk daftar yang besar atau hampir terurut, tetapi mudah dipahami dan diterapkan.

2. Selection Sort

Selection Sort adalah algoritma pengurutan sederhana yang berulang kali menemukan elemen minimum (atau maksimum) dari bagian daftar yang tidak diurutkan dan menukarnya dengan elemen yang tidak diurutkan paling kiri. Proses ini diulang hingga seluruh daftar diurutkan.



Gambar algoritma pada Selection sort

Berikut implementasi pada Selection sort:

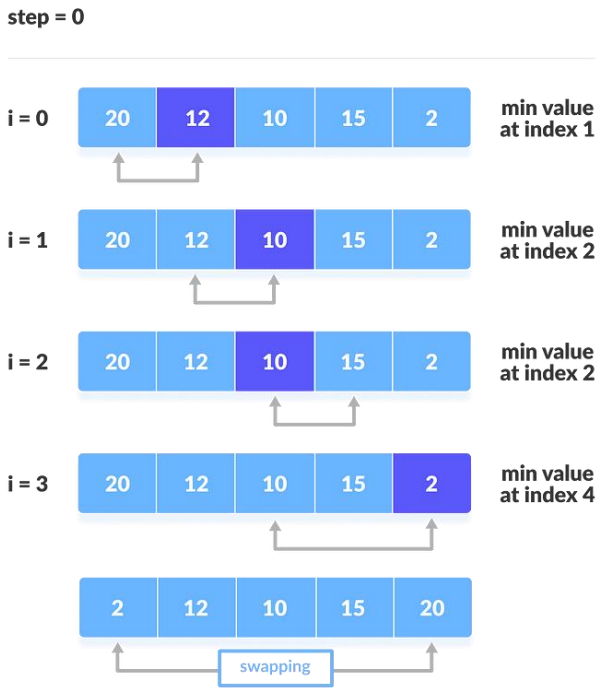
```
func  seleksi (array [] int , ukuran int ) {
  untuk i := 0 ; i < ukuran; i++ {
  min := i
   untuk j := i + 1 ; j < ukuran; j++ {
    jika array [min] > array [j] {
    min = j
   }
  }
  temp := array [min]
  array [min] = array [i]
  array [i] = temp
}
fmt.Println( "Array yang diurutkan berdasarkan pilihan
adalah: " )
fmt.Println(array)
}
```

Pada fungsi ini **selection** mengambil dua parameter yaitu **array**, yang merupakan potongan bilangan bulat yang akan diurutkan, dan **size**, yang menyatakan ukuran array. Perulangan luar **for** berulang dari 0 hingga **size-1** untuk memilih elemen minimum saat ini. Di dalam loop luar, variabel **min** diinisialisasi dengan nilai **i**, yang menunjukkan indeks elemen minimum saat ini. Perulangan dalam **for** dimulai dari **i+1** dan berlanjut hingga **size-1**. Perulangan ini membandingkan setiap elemen di bagian array yang belum diurutkan dengan elemen minimum saat ini (**array[min]**). Jika suatu elemen pada indeks **j** ditemukan lebih kecil dari elemen minimum saat ini (**array[min]**), **min** indeks diperbarui menjadi **j**. Proses ini menemukan elemen terkecil di bagian array yang tidak diurutkan.

Setelah loop bagian dalam selesai, elemen minimum (**array[min]**) ditukar dengan elemen pada indeks saat ini (**array[i]**). Ini menempatkan elemen terkecil pada posisi yang diurutkan dengan benar. Proses ini berulang untuk iterasi berikutnya dari loop luar, memilih elemen minimum berikutnya dan menempatkannya pada posisi yang sesuai. Setelah putaran luar selesai, seluruh susunan diurutkan dalam urutan menaik. Terakhir, array yang diurutkan dicetak menggunakan **fmt.Println** fungsi tersebut. Selection Sort memiliki kompleksitas waktu $O(n^2)$ dalam kasus terburuk dan rata-rata, di mana n adalah jumlah elemen dalam daftar. Ia berkinerja lebih baik daripada Bubble Sort dalam hal jumlah pertukaran tetapi masih memiliki keterbatasan untuk daftar yang lebih besar.

3. Insertion Sort

Insertion sort adalah algoritma pengurutan sederhana lainnya yang bekerja dengan membagi array menjadi dua bagian: bagian yang sudah diurutkan dan bagian yang belum diurutkan. Algoritma ini akan mengulangi bagian yang belum diurutkan dan secara bertahap memasukkan setiap elemen ke posisi yang benar di bagian yang sudah diurutkan.



Gambar algortima pada Insertion sort

Berikut implementasi pada Insertion sort:

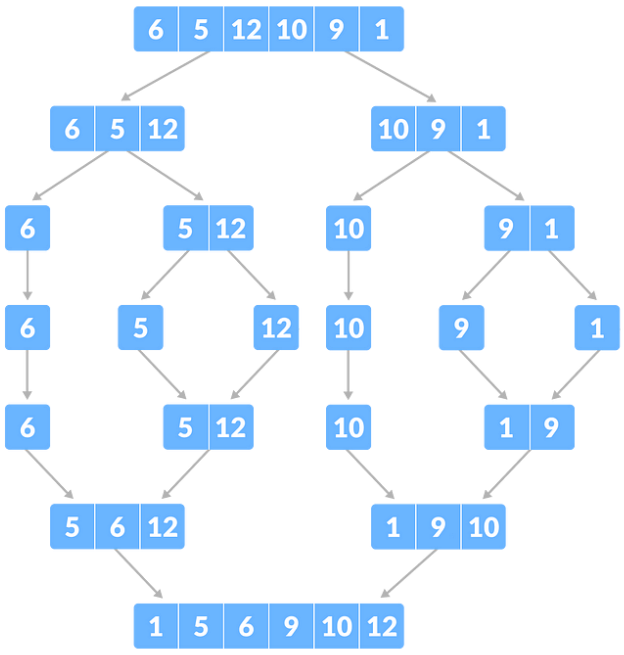
```
func  insertion (array [] int , size int ) {
     untuk i, v := range array {
        j := i - 1
        jika i >= 1 {
             untuk j >= 0 dan array[j] > v {
                array[j+ 1 ] = array[j]
                j--
            }
            array[j+ 1 ] = v
        }
    }
    fmt.Println( "Array yang diurutkan dengan penyisipan
adalah : " )
    fmt.Println(array)
}
```

Fungsi ini insertionmengambil dua parameter yaitu **array**, yang merupakan potongan bilangan bulat yang akan diurutkan, dan **size**, yang menyatakan ukuran array. Loop luar **for** mengiterasi setiap elemen array menggunakan **range** kata kunci. Untuk setiap iterasi, loop ini menetapkan indeks saat ini ke **i** dan nilai elemen terkait ke **v**. Di dalam loop, sebuah variabel **j** diinisialisasi dengan nilai **i - 1**. Variabel ini melacak posisi di bagian yang diurutkan tempat elemen saat ini **v** perlu disisipkan. Kondisi ini **if i >= 1** memeriksa apakah elemen saat ini bukan elemen pertama dari array. Kondisi ini memastikan bahwa ada posisi yang valid untuk memasukkan elemen.

Di dalam loop bagian dalam **for**, ia membandingkan elemen saat ini **v** dengan elemen di sebelah kirinya (**array[j]**). Ia terus menggeser elemen yang lebih besar ke kanan sebanyak satu posisi hingga menemukan posisi yang tepat untuk memasukkan elemen saat ini. Perulangan berlanjut hingga mencapai awal array (**j >= 0**) atau hingga menemukan elemen yang lebih kecil atau sama dengan elemen saat ini (**array[j] <= v**). Setelah menemukan posisi yang benar, elemen saat ini **v** dimasukkan ke dalam bagian array yang telah diurutkan pada indeks **j+1**. Loop luar melanjutkan proses ini untuk setiap elemen array, secara bertahap membangun bagian yang telah diurutkan. Terakhir, array yang diurutkan dicetak menggunakan **fmt.Println** fungsi tersebut. Insertion sort memiliki kompleksitas waktu O(n^2), yang membuatnya efisien untuk array kecil dan array yang diurutkan sebagian. Namun, untuk array besar, algoritma pengurutan lain seperti merge sort atau quicksort umumnya lebih efisien.

4. Merge Sort

Merge sort merupakan algoritma pengurutan bagi-dan-kuasai yang bekerja dengan membagi array secara rekursif menjadi bagian-bagian yang lebih kecil, mengurutkannya satu per satu, lalu menggabungkan kembali bagian-bagian yang telah diurutkan tersebut untuk memperoleh array terurut akhir.



Gambar algoritma pada Merge sort

Berikut implementasi pada Merge sort:

```
func  mergeSort (array [] int ) [] int {
    panjang := len (array)
     jika panjang <= 1 {
         kembalikan array
    }
```

```
    tengah := panjang / 2
     kiri := mergeSort (array [: tengah])
    kanan := mergeSort (array [tengah:])

     kembalikan gabungan (kiri, kanan)
}

 func  merge (kiri, kanan [] int ) [] int {
    digabungkan := buat ([] int , 0 , len (kiri) + len
(kanan))
    i, j := 0 , 0

    untuk i < len (kiri) dan j < len (kanan) {
         jika kiri [i] < kanan [j] {
            digabungkan = tambahkan (digabungkan, kiri
[i])
            i++
        } yang lain {
            digabungkan = tambahkan (digabungkan, kanan
[j])
            j++
        }
    }

    digabungkan = tambahkan (digabungkan, kiri [i:] ...)
    digabungkan = tambahkan (digabungkan, kanan [j:]
...)

     kembalikan gabungan
}
```

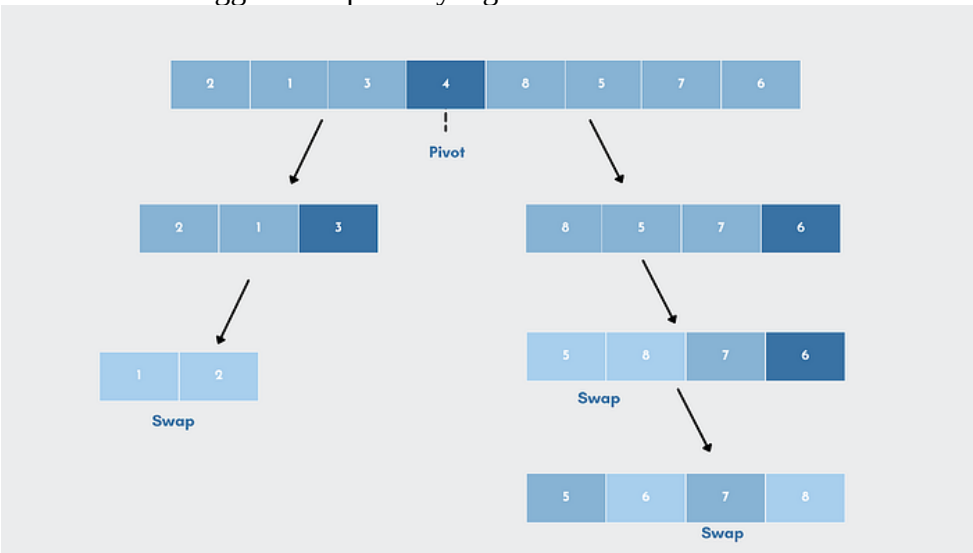
Fungsi ini **mergeSort** adalah titik masuk untuk algoritma pengurutan gabungan. Fungsi ini mengambil array **array** sebagai input dan mengembalikan array yang telah diurutkan. Panjang array diperoleh menggunakan **len(array)**. Kasus dasar rekursi merge sort ditangani di awal fungsi **mergeSort**. Jika panjang array kurang dari atau sama dengan 1, berarti array sudah diurutkan, jadi fungsi tersebut hanya mengembalikan array apa adanya. Jika array memiliki lebih dari satu elemen, fungsi tersebut akan membagi array menjadi dua bagian. Titik tengah dihitung dengan membagi panjang array dengan 2 menggunakan **mid := length / 2**. Setengah bagian kiri array diperoleh dengan mengiris **array** dari awal hingga titik tengah menggunakan **array[:mid]**, dan setengah bagian kanan diperoleh dengan mengiris dari titik tengah hingga akhir menggunakan **array[mid:]**. Bagian kiri dan kanan diurutkan secara rekursif dengan memanggil **mergeSort** masing-masing bagian, menghasilkan dua array yang diurutkan yaitu **left** dan **right**. Bagian-bagian yang diurutkan kemudian digabungkan dengan memanggil **merge** fungsi tersebut. **merge** Fungsi tersebut mengambil array **left**

dan **right** sebagai input dan mengembalikan array baru yang digabungkan dan diurutkan. Fungsi ini **merge** menginisialisasi irisan kosong **merged** untuk menyimpan array yang digabung dan diurutkan. Fungsi ini juga menginisialisasi dua indeks **i** dan **j** untuk melacak posisi saat ini dalam array **left** dan **right**. Proses penggabungan dimulai dengan perulangan yang membandingkan elemen dari array **left** dan **right**. Perulangan berlanjut selama panjang **i** dan **j** kurang dari panjang array masing-masing. Di dalam loop, algoritma membandingkan elemen pada indeks **i** dan **j**. Jika elemen di **left[i]** lebih kecil dari elemen di **right[j]**, elemen tersebut ditambahkan ke **merged** array, dan **i** bertambah. Jika tidak, elemen di **right[j]** ditambahkan ke **merged** array, dan **j** bertambah.

Perulangan berlanjut hingga salah satu array telah dilintasi sepenuhnya. Pada titik tersebut, elemen yang tersisa di array lainnya telah diurutkan dan dapat langsung ditambahkan ke array **merged** menggunakan **append** fungsi dan notasi pemotongan (**merged = append(merged, left[i:]...**) dan **merged = append(merged, right[j:]...)**). Akhirnya, **merged** array yang berisi elemen-elemen yang diurutkan dari kedua bagian dikembalikan sebagai hasil operasi penggabungan. Dengan membagi array secara rekursif menjadi bagian-bagian yang lebih kecil dan menggabungkannya kembali, merge sort secara efektif mengurutkan array dalam urutan menaik. Kompleksitas waktu merge sort adalah O(n log n), sehingga efisien untuk mengurutkan array yang besar. Ini adalah algoritma pengurutan yang stabil, artinya ia mempertahankan urutan relatif elemen yang sama dalam array yang diurutkan.

5. Quick Sort

Quick sort adalah algoritma sorting lain yang efisien dengan metode bagi-dan-kuasai. Algoritme ini bekerja dengan memilih elemen pivot dari array dan membagi elemen lainnya menjadi dua sub-array berdasarkan apakah elemen tersebut lebih kecil atau lebih besar dari pivot. Sub-array kemudian diurutkan secara rekursif menggunakan proses yang sama.



Gambar Algoritma pada Quick sort

Berikut penjelasan cara kerja quick sort :

1. Algoritme memilih elemen pivot dari array. Pivot dapat dipilih dengan berbagai cara, seperti memilih elemen pertama, terakhir, atau tengah dari array.
2. Algoritme membagi array menjadi dua sub-array: satu berisi elemen yang lebih kecil dari pivot dan yang lainnya berisi elemen yang lebih besar dari pivot. Hal ini dilakukan dengan melakukan iterasi melalui array dan memindahkan elemen yang lebih kecil dari pivot ke kiri dan elemen yang lebih besar dari pivot ke kanan.
3. Setelah langkah pemartisian, elemen pivot berada pada posisi akhir yang diurutkan. Elemen di sebelah kiri pivot lebih kecil, dan elemen di sebelah kanan lebih besar.
4. Algoritme kemudian menerapkan proses yang sama secara rekursif ke subarray di kedua sisi pivot. Hal ini dilakukan hingga setiap subarray berisi nol atau satu elemen, karena subarray tersebut sudah dianggap terurut.
5. Terakhir, subarray digabungkan untuk memperoleh array terurut akhir.

Berikut implementasi pada Quick sort :

```
func  quickSort (array [] int ) [] int {
     jika  len (array) <= 1 {
         kembalikan array
    }

    pivot := array[ len (array) -1 ]
     var kiri, kanan [] int

    untuk i := 0 ; i < len (array) -1 ; i++ {
         jika array[i] <= pivot {
            kiri = tambahkan (kiri, array[i])
        } yang lain {
            kanan = tambahkan (kanan, array[i])
        }
    }

    kiri = quickSort(kiri)
    kanan = quickSort(kanan)

     kembalikan  tambahkan ( tambahkan (kiri, pivot),
kanan...)
}
```

Pada fungsi **quickSort** mengambil serangkaian bilangan bulat sebagai input dan mengembalikan array yang diurutkan menggunakan algoritma pengurutan cepat. Kasus dasar rekursi ditangani di awal fungsi. Jika panjang array kurang dari atau sama dengan 1, berarti array sudah diurutkan, jadi fungsi hanya mengembalikan array apa adanya. Elemen pivot dipilih sebagai elemen terakhir array (**pivot := array[len(array)-1]**). Algoritma kemudian mengulangi array (tidak termasuk pivot) dan membagi elemen menjadi dua sub-array yaitu **left** dan **right**. Elemen yang lebih kecil dari atau sama dengan pivot ditambahkan

ke **left** sub-array, sedangkan elemen yang lebih besar dari pivot ditambahkan ke **right** sub-array. Fungsi ini **quickSort** dipanggil secara rekursif pada subarray **left** dan **right.** Terakhir, subarray digabungkan dengan menambahkan subarray **left**, diikuti oleh elemen pivot, lalu subarray **right**. Array gabungan yang dihasilkan dikembalikan sebagai hasil yang diurutkan. Kompleksitas waktu quick sort adalah O(n log n) secara rata-rata, tetapi dapat menurun hingga $O(n^2)$ dalam kasus terburuk. Namun, skenario terburuk jarang terjadi dan dapat dikurangi dengan menggunakan berbagai strategi pemilihan pivot dan teknik partisi.

II. GUIDED

1).

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.

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()
    }
}
```

SCREENSHOT OUTPUT :

```
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11> go run
modul11\Guided\Guided1.go"
Masukkan jumlah daerah (n): 3
Masukkan jumlah rumah kerabat untuk daerah ke-1: 4
Masukkan nomor rumah kerabat untuk daerah ke-1: 1 2 3 4
Hasil urutan rumah untuk daerah ke-1: 4 3 2 1
Masukkan jumlah rumah kerabat untuk daerah ke-2: 5
Masukkan nomor rumah kerabat untuk daerah ke-2: 2 3 1 1 1
Hasil urutan rumah untuk daerah ke-2: 3 2 1 1 1
Masukkan jumlah rumah kerabat untuk daerah ke-3: 5
Masukkan nomor rumah kerabat untuk daerah ke-3: 0 2 0 7 5
Hasil urutan rumah untuk daerah ke-3: 7 5 2 0 0
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11>
```

PENJELASAN PROGRAM :

Program tersebut adalah program mengurutkan data berupa nomor rumah kerabat dari beberapa daerah menggunakan selection sort. Pengguna diminta memasukkan jumlah daerah, kemudian untuk setiap daerah diminta jumlah rumah kerabat serta nomor rumahnya. Nomor-nomor rumah tersebut disimpan dalam array dan diurutkan menggunakan fungsi selection sort dengan mencari elemen terbesar dari array yang belum diurutkan, kemudian menukarnya ke posisi yang sesuai. Setelah proses pengurutan selesai, program mencetak hasil nomor rumah yang telah diurutkan untuk setiap daerah. Pengecekan input dilakukan untuk memastikan bahwa jumlah daerah dan rumah berada dalam posisi yang tepat.

2).

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

Masukan terdiri dari sekumpulan bilangan bulat yang diakhiri 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

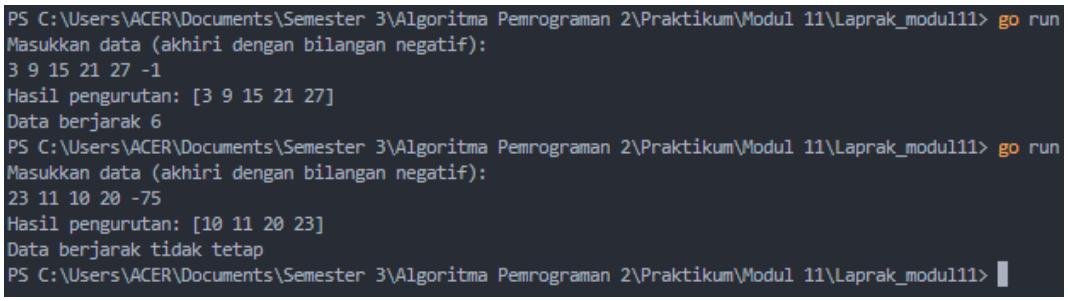
No	Masukan	Keluaran
1	31 13 25 43 1 7 19 37 -5	1 7 13 19 25 31 37 43 Data berjarak 6
2	4 40 14 8 26 1 38 2 32 -31	1 2 4 8 14 26 32 38 40 Data berjarak tidak tetap

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
}

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")
    }
}
```

SCREENSHOT OUTPUT :



PENJELASAN PROGRAM :

Program tersebut adalah program untuk membaca data integer kemudian diurutkan menggunakan insertion sort dan memeriksa apakah data yang terurut berjarak sama terhadap data sebelumnya. Pengguna diminta menginputkan serangkaian bilangan integer yang akan disimpan dalam array hingga input negatif diberikan sebagai tanda akhir. Setelah itu array diurutkan menggunakan fungsi insertionSort dengan cara membandingkan elemen saat ini dengan elemen-elemen sebelumnya, lalu menyisipkannya di posisi yang sesuai, sehingga elemen-elemen sebelum posisi tersebut selalu terurut. Setelah array diurutkan fungsi isDataConsistentlySpaced akan memeriksa apakah selisih antar elemen dalam array terurut tersebut tetap (konsisten). Pemeriksaan ini dilakukan dengan menghitung selisih absolut antara elemen pertama dan kedua sebagai referensi, lalu membandingkannya dengan selisih antar elemen lainnya. Jika ada selisih yang berbeda, array dianggap tidak memiliki jarak yang konsisten. Pada hasil dari program ini mencakup tampilan array yang telah diurutkan serta informasi tentang apakah elemen-elemen array berjarak tetap, termasuk nilai selisih tersebut jika konsisten ataupun sebaliknya.

III. UNGUIDED

1.

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 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 pencetakan, mulai dengan mencetak semua nilai ganjil lebih dulu, kemudian setelah selesai cetaklah semua nilai genapnya.

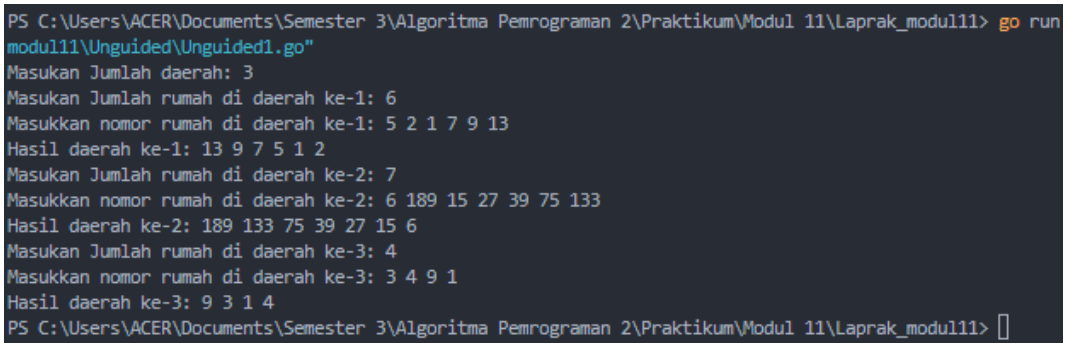
SOURCE CODE

```go
package main

import (
    "fmt"
)

func main() {
    var nDaerah int
    fmt.Print("Masukan Jumlah daerah: ")
    fmt.Scan(&nDaerah)

    if nDaerah <= 0 {
        fmt.Println("Error: Jumlah daerah harus lebih dari 0")
        return
    }
```

```
    for i := 1; i <= nDaerah; i++ {
        var nRumah int
        fmt.Printf("Masukan Jumlah rumah di daerah %d: ", i)
        fmt.Scan(&nRumah)

        if nRumah <= 0 {
            fmt.Println("Jumlah rumah harus positif!")
            return
        }

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

        ganjil := []int{}
        genap := []int{}
        for _, x := range dataRumah {
            if x%2 == 0 {
                genap = append(genap, x)
            } else {
                ganjil = append(ganjil, x)
            }
        }
        sortSelection(ganjil, false)
        sortSelection(genap, true)

        fmt.Printf("Hasil daerah %d: ", i)
        for _, g := range ganjil {
            fmt.Printf("%d ", g)
        }
        for _, g := range genap {
            fmt.Printf("%d ", g)
        }
        fmt.Println()
    }
}

func sortSelection(arr []int, ascending bool) {
    for i := 0; i < len(arr)-1; i++ {
        idx := i
        for j := i + 1; j < len(arr); j++ {
            if (ascending && arr[j] < arr[idx]) || (!ascending
&& arr[j] > arr[idx]) {
```

```
                idx = j
            }
        }
        arr[i], arr[idx] = arr[idx], arr[i]
    }
}
```

SCREENSHOT OUTPUT



PENJELASAN PROGRAM

Program tersebut adalah program mengelompokkan dan mengurutkan nomor rumah di beberapa daerah berdasarkan bilangan ganjil dan genap. Pertama, program meminta pengguna memasukkan jumlah daerah (nDaerah). Jika jumlah daerah ≤ 0, program akan menghentikan eksekusi dengan menampilkan pesan kesalahan. kemudian, untuk setiap daerah, pengguna diminta memasukkan jumlah rumah (nRumah) dan daftar nomor rumah. Nomor rumah tersebut kemudian dikelompokkan menjadi dua bagian yaitu bilangan ganjil dan bilangan genap. Nomor-nomor ganjil diurutkan secara menurun (descending), sedangkan nomor-nomor genap diurutkan secara menaik (ascending) menggunakan algoritma Selection Sort, yang digunakan dalam fungsi sortSelection. Pada hasil outputnya menampilkan nomor rumah ganjil terlebih dahulu disusul oleh nomor rumah genap untuk setiap daerah.

2).

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.

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.

SOURCE CODE

```
package main

import (
    "fmt"
)

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

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

    selectionSort(array)

    if jumlah%2 == 1 {
        return float64(array[jumlah/2])
    }
    tengah1 := array[(jumlah/2)-1]
    tengah2 := array[jumlah/2]
    return float64(tengah1+tengah2) / 2.0
}

func main() {
    var jumlahData int
    fmt.Print("Masukkan jumlah elemen data: ")
    fmt.Scan(&jumlahData)

    if jumlahData <= 0 || jumlahData > 1000000 {
        fmt.Println("Jumlah elemen harus lebih besar dari 0 dan
tidak boleh lebih dari 1.000.000.")
        return
    }
```

```
    data := make([]int, jumlahData)
    fmt.Printf("Masukkan %d elemen data:\n", jumlahData)
    for i := 0; i < jumlahData; i++ {
        fmt.Scan(&data[i])
    }
    median := hitungMedian(data)
    fmt.Printf("Median dari data adalah: %.2f\n", median)
}
```

SCREENSHOT OUTPUT

```
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11> go run
modul11\Unguided\Unguided2.go"
Masukkan jumlah elemen data: 3
Masukkan 3 elemen data:
7 11 23
Median dari data adalah: 11.00
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11> go run
modul11\Unguided\Unguided2.go"
Masukkan jumlah elemen data: 10
Masukkan 10 elemen data:
7 23 11 5 19 2 29 3 13 17
Median dari data adalah: 12.00
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11>
```

PENJELASAN PROGRAM

Program tersebut adalah program untuk menghitung median dari kumpulan angka yang diinputkankan oleh pengguna. Pertama, program meminta pengguna untuk menentukan jumlah angka yang ingin dimasukkan dengan batasan bahwa jumlah tersebut harus lebih besar dari nol dan tidak lebih dari satu juta. Jika input tidak valid, program menampilkan pesan error dan keluar. Selanjutnya, program mengumpulkan data dalam bentuk array dan menggunakan algoritma selection sort untuk mengurutkan angka-angka tersebut. Setelah diurutkan, fungsi hitungMedian menghitung median berdasarkan jumlah elemen. jika mediannya ganjil, maka nilai mediannya berada dielemen tengah, sedangkan jika mediannya genap, maka mediannya dihitung sebagai rata-rata dari dua elemen tengah. Pada akhirnya, program menampilkan hasil output median kepada pengguna dengan format dua angka desimal, sehingga memberikan gambaran yang jelas mengenai nilai tengah dari data yang diinput.

3). 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.}
```

SOURCE CODE

```
package main

import (
    "fmt"
    "sort"
)

type Buku struct {
    Judul, Penulis, Penerbit string
    Tahun, Rating            int
}

func DaftarkanBuku(n int) []Buku {
    pustaka := make([]Buku, n)
    for i := 0; i < n; i++ {
        fmt.Printf("Masukkan data buku ke-%d (Judul Penulis
Penerbit Tahun Rating):\n", i+1)
        fmt.Scan(&pustaka[i].Judul, &pustaka[i].Penulis,
&pustaka[i].Penerbit, &pustaka[i].Tahun, &pustaka[i].Rating)
    }
    return pustaka
}

func CetakTefavorit(pustaka []Buku) {
    if len(pustaka) == 0 {
        fmt.Println("Tidak ada data buku.")
        return
    }
    tertinggi := pustaka[0]
    for _, buku := range pustaka {
        if buku.Rating > tertinggi.Rating {
            tertinggi = buku
        }
    }
    fmt.Println("Buku dengan rating tertinggi:")
    fmt.Printf("%s oleh %s, diterbitkan oleh %s (%d) - Rating:
%d\n", tertinggi.Judul, tertinggi.Penulis, tertinggi.Penerbit,
tertinggi.Tahun, tertinggi.Rating)
}

func UrutkanBuku(pustaka []Buku) {
    sort.Slice(pustaka, func(i, j int) bool {
        return pustaka[i].Rating > pustaka[j].Rating
    })
}
```

```
func Cetak5Terbaik(pustaka []Buku) {
    fmt.Println("5 Buku dengan rating tertinggi:")
    for i := 0; i < len(pustaka) && i < 5; i++ {
        fmt.Printf("%s oleh %s (%d) - Rating: %d\n",
pustaka[i].Judul, pustaka[i].Penulis, pustaka[i].Tahun,
pustaka[i].Rating)
    }
}

func CariBuku(pustaka []Buku, rating int) {
    for _, buku := range pustaka {
        if buku.Rating == rating {
            fmt.Println("Buku ditemukan:")
            fmt.Printf("%s oleh %s, diterbitkan oleh %s (%d) -
Rating: %d\n", buku.Judul, buku.Penulis, buku.Penerbit,
buku.Tahun, buku.Rating)
            return
        }
    }
    fmt.Println("Tidak ada buku dengan rating seperti itu.")
}

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

    if n <= 0 {
        fmt.Println("Jumlah buku harus lebih besar dari nol.")
        return
    }
    pustaka := DaftarkanBuku(n)

    UrutkanBuku(pustaka)
    CetakTefavorit(pustaka)
    Cetak5Terbaik(pustaka)

    var rating int
    fmt.Print("Masukkan rating untuk mencari buku: ")
    fmt.Scan(&rating)
    CariBuku(pustaka, rating)
}
```

SCREENSHOT OUTPUT

```
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11> go run
3.go"
Masukkan jumlah buku: 6
Masukkan data buku ke-1 (Judul Penulis Penerbit Tahun Rating):
HarryPotter JKRowling Bloomsbury 1997 5
Masukkan data buku ke-2 (Judul Penulis Penerbit Tahun Rating):
GoProgramming AlanDonovan AddisonWesley 2016 6
Masukkan data buku ke-3 (Judul Penulis Penerbit Tahun Rating):
Twilight Meyer Gramedia 2005 7
Masukkan data buku ke-4 (Judul Penulis Penerbit Tahun Rating):
Matahari Tere Gramedia 2016 4
Masukkan data buku ke-5 (Judul Penulis Penerbit Tahun Rating):
Sunset Alvi Mizan 2017 6
Masukkan data buku ke-6 (Judul Penulis Penerbit Tahun Rating):
Aroma Dee Bentang 2012 5
Buku dengan rating tertinggi:
Twilight oleh Meyer, diterbitkan oleh Gramedia (2005) - Rating: 7
5 Buku dengan rating tertinggi:
Twilight oleh Meyer (2005) - Rating: 7
GoProgramming oleh AlanDonovan (2016) - Rating: 6
Sunset oleh Alvi (2017) - Rating: 6
HarryPotter oleh JKRowling (1997) - Rating: 5
Aroma oleh Dee (2012) - Rating: 5
Masukkan rating untuk mencari buku: 6
Buku ditemukan:
GoProgramming oleh AlanDonovan, diterbitkan oleh AddisonWesley (2016) - Rating: 6
PS C:\Users\ACER\Documents\Semester 3\Algoritma Pemrograman 2\Praktikum\Modul 11\Laprak_modul11>
```

PENJELASAN PROGRAM

Program di atas adalah program mengelola data buku. Program dimulai dengan pengguna diminta menginputkan jumlah buku yang ingin didaftarkan seperti judul, penulis, penerbit, tahun, dan rating, lalu diinput melalui fungsi DaftarkanBuku dan disimpan dalam sebuah slice. Setelah semua data terkumpul, fungsi UrutkanBuku mengurutkan daftar buku berdasarkan rating secara menurun, sehingga buku dengan rating tertinggi berada di urutan pertama. Program kemudian menampilkan buku dengan rating tertinggi menggunakan fungsi CetakTefavorit dan mencetak hingga lima buku terbaik berdasarkan rating melalui fungsi Cetak5Terbaik. Selain itu, pengguna dapat mencari buku berdasarkan rating tertentu dengan fungsi CariBuku, yang akan menampilkan detail buku jika ditemukan atau memberikan pesan jika tidak ada buku yang cocok. Semua fungsi program tersebut akan berada dalam fungsi utama main yang mengelola input, pengolahan data, serta penampilan hasil output.