LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL 12 & 13

PENGURUTAN DATA

Disusun Oleh :

Ariiq Radhitya Pradana / 2311102260

IF 11 06

Dosen Pengampu :

ABEDNEGO DWI SEPTIADI

PROGRAM STUDI S1 TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

TELKOM UNIVERSITY PURWOKERTO

2024

I. DASAR TEORI

Pengurutan data, atau sorting, adalah proses penting dalam pemrograman yang bertujuan untuk mengatur elemen-elemen dalam suatu kumpulan data ke dalam urutan tertentu, baik secara menaik (ascending) maupun menurun (descending). Dalam bahasa pemrograman Go (Golang), terdapat berbagai algoritma pengurutan yang dapat digunakan, masing-masing dengan karakteristik dan efisiensi yang berbeda. Beberapa algoritma pengurutan yang umum digunakan antara lain Bubble Sort, Insertion Sort, Merge Sort, Quick Sort, dan Bucket Sort.

Bubble Sort adalah salah satu algoritma paling sederhana yang bekerja dengan membandingkan elemen bersebelahan dan menukarnya jika berada dalam urutan yang salah. Proses ini diulang hingga tidak ada lagi penukaran yang diperlukan. Meskipun mudah dipahami dan diimplementasikan, Bubble Sort memiliki kompleksitas waktu $O(n^2)$ pada kasus rata-rata dan terburuk, menjadikannya kurang efisien untuk dataset besar. Di sisi lain, Insertion Sort membangun urutan terurut secara bertahap dengan menyisipkan elemen baru ke dalam posisi yang tepat. Algoritma ini juga memiliki kompleksitas $O(n^2)$, tetapi sering kali lebih cepat dibandingkan Bubble Sort pada dataset kecil karena lebih sedikit perbandingan yang dilakukan.

Merge Sort adalah algoritma pengurutan yang lebih efisien dengan menggunakan pendekatan "divide and conquer". Algoritma ini membagi array menjadi dua bagian, mengurutkan masing-masing secara rekursif, dan kemudian menggabungkannya kembali. Dengan kompleksitas waktu $O(n \log n)$ pada semua kasus, Merge Sort sangat cocok untuk pengurutan dataset besar. Quick Sort juga menggunakan metode "divide and conquer" dengan memilih elemen sebagai pivot dan mengatur ulang elemen di sekitarnya berdasarkan nilai relatif terhadap pivot. Meskipun Quick Sort memiliki kompleksitas waktu $O(n \log n)$ pada kasus rata-rata, ia dapat mencapai $O(n^2)$ dalam kasus terburuk jika pivot dipilih secara tidak optimal.

Dalam praktiknya, Go menyediakan paket standar `sort` yang memudahkan implementasi algoritma pengurutan. Misalnya, fungsi `sort.Ints` dapat digunakan untuk mengurutkan slice integer dengan sangat mudah. Contoh implementasi sederhana menunjukkan bagaimana slice dapat diurutkan dalam urutan menaik hanya dengan satu baris kode. Secara keseluruhan, pemilihan algoritma pengurutan yang tepat sangat penting untuk meningkatkan efisiensi dan kinerja aplikasi, tergantung pada karakteristik data yang akan diurutkan serta kebutuhan spesifik dari program tersebut.

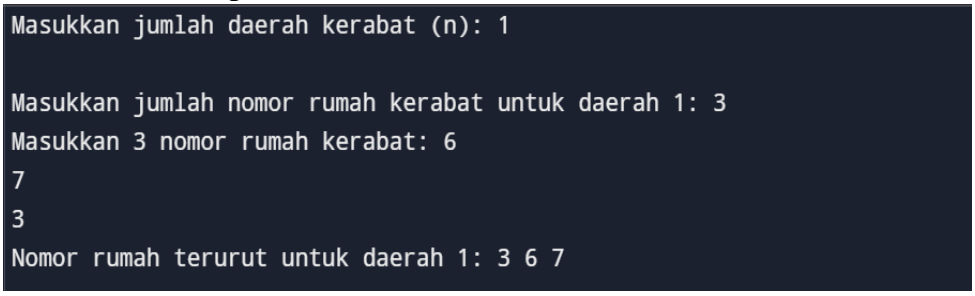
II. GUIDED

Guided 1

```
package main

import (
    "fmt"
)

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

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

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

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

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

        // Tampilkan hasil
        fmt.Printf("Nomor rumah terurut untuk daerah %d:
", daerah)
```

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

Screenshoot Output



Deskripsi Program

Kode Go di atas adalah program untuk mengurutkan daftar nomor rumah kerabat menggunakan algoritma **Selection Sort**. Program ini dimulai dengan meminta pengguna untuk memasukkan jumlah daerah kerabat yang akan diproses. Untuk setiap daerah, pengguna diminta untuk memasukkan jumlah rumah kerabat beserta nomor-nomor rumahnya. Nomor-nomor rumah tersebut disimpan dalam sebuah array.
Program kemudian menggunakan fungsi selectionSort untuk mengurutkan array tersebut. Fungsi ini bekerja dengan menemukan elemen terkecil dari array yang belum terurut dan menukarnya dengan elemen di posisi awal yang sedang diproses. Proses ini diulang hingga semua elemen array terurut.

Setelah proses pengurutan selesai, program mencetak daftar nomor rumah yang sudah terurut untuk setiap daerah. Hasilnya adalah nomor rumah yang terurut dari nilai terkecil hingga terbesar, dan ini dilakukan secara independen untuk setiap daerah. Program menggunakan input berbasis konsol untuk menerima data dari pengguna dan memberikan keluaran berupa daftar nomor rumah yang terurut.

Guided 2

```
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

```
Masukkan data integer (akhiri dengan bilangan negatif):
3 4 7 9 5 -1
Array setelah diurutkan:
3 4 5 7 9
Data berjarak tidak tetap
```

Deskripsi Program

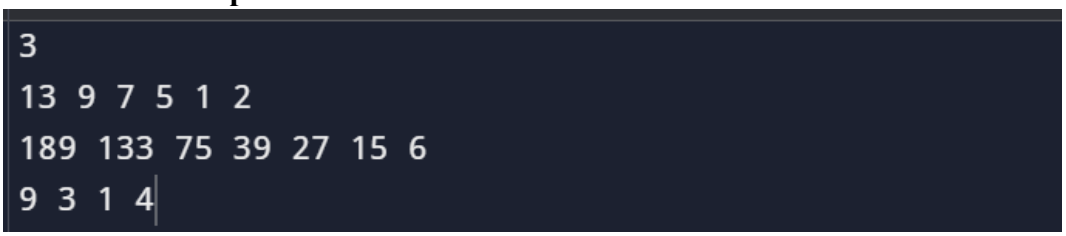
Program ini ditulis dalam bahasa Go untuk menerima input berupa bilangan bulat dari pengguna, mengurutkannya, dan menentukan apakah selisih antara elemen-elemen yang berdekatan dalam array hasil pengurutan bersifat tetap (konstan). Pengguna diminta memasukkan bilangan bulat satu per satu hingga bilangan negatif dimasukkan sebagai penanda akhir input, dengan bilangan negatif itu sendiri tidak dimasukkan ke dalam array. Setelah itu, array diurutkan menggunakan algoritma *Insertion Sort*, yang menyusun elemen-elemen dalam urutan menaik. Program kemudian memeriksa apakah selisih antara elemen-elemen yang berdekatan tetap konstan. Jika selisih tetap, program menampilkan nilai selisih tersebut; jika tidak, program menyatakan bahwa selisihnya tidak tetap. Terakhir, program menampilkan array hasil pengurutan beserta status jaraknya. Program ini membantu menganalisis pola dalam data numerik, terutama dalam mendeteksi barisan aritmetika setelah pengurutan.

III. UNGUIDED

Unguided 1

```
package main
import (
    "fmt"
    "sort"
    "strings"
)

func main() {
    // Input data sesuai format tabel
    input := [][]int{
        {3},
        {5, 2, 1, 7, 9, 13},
        {6, 189, 15, 27, 39, 75, 133},
        {3, 4, 9, 1},
    }

    // Proses untuk setiap daerah
    for _, nums := range input {
        oddNumbers := []int{}
        evenNumbers := []int{}

        // Pisahkan bilangan ganjil dan genap
        for _, num := range nums {
            if num%2 == 0 {
                evenNumbers = append(evenNumbers, num)
            } else {
                oddNumbers = append(oddNumbers, num)
            }
        }

        // Urutkan ganjil dari besar ke kecil
        sort.Sort(sort.Reverse(sort.IntSlice(oddNumbers)
))

        // Urutkan genap dari kecil ke besar
        sort.Ints(evenNumbers)

        // Gabungkan hasilnya
        output := append(oddNumbers, evenNumbers...)

        // Cetak hasil dalam satu baris
        outputStr :=
strings.Trim(strings.Replace(fmt.Sprint(output), " ", "
", -1), "[]")
        fmt.Println(outputStr)
    }
}
```

Screenshoot Output



Deskripsi Program

Kode di atas ditulis dalam bahasa Go dan bertujuan untuk memproses beberapa set angka, memisahkan angka-angka tersebut menjadi bilangan ganjil dan genap, lalu mengurutkannya berdasarkan kriteria tertentu sebelum mencetak hasilnya. Data input disusun sebagai array dua dimensi, di mana setiap elemen merepresentasikan sebuah daerah dengan sejumlah angka.

Untuk setiap elemen dalam input, program memisahkan bilangan ganjil dan genap. Bilangan ganjil diurutkan secara menurun (dari besar ke kecil) menggunakan fungsi sort.Sort dengan sort.Reverse(sort.IntSlice), sementara bilangan genap diurutkan secara menaik (dari kecil ke besar) menggunakan sort.Ints. Kedua daftar hasil tersebut kemudian digabungkan dalam satu array, di mana bilangan ganjil berada lebih dulu diikuti oleh bilangan genap.

Akhirnya, array hasil gabungan tersebut diubah menjadi string yang rapi menggunakan fmt.Sprint, dengan elemen-elemen yang dipisahkan oleh spasi, lalu dicetak dalam satu baris untuk setiap daerah dalam input. Hasil akhir memberikan representasi yang terurut sesuai aturan yang ditentukan untuk setiap set angka dalam input.

Unguided 2

```
package main

import (
	"bufio"
	"fmt"
	"os"
	"sort"
	"strconv"
	"strings"
)

// Fungsi untuk menghitung median dari data yang telah
diurutkan
func calculateMedian(data []int) int {
	n := len(data)
	if n%2 == 0 {
		return (data[n/2-1] + data[n/2]) / 2
	}
	return data[n/2]
}

func main() {
	reader := bufio.NewReader(os.Stdin)

	fmt.Println("Masukkan deretan angka, pisahkan dengan
spasi. Akhiri dengan -5313:")

	// Membaca input dari pengguna
	input, _ := reader.ReadString('\n')
	input = strings.TrimSpace(input)

	// Memisahkan input menjadi angka
	inputs := strings.Split(input, " ")

	var data []int

	for _, item := range inputs {
		num, err := strconv.Atoi(item)
		if err != nil {
			fmt.Println("Input tidak valid:", item)
			return
		}

		// Jika input adalah -5313, hentikan program
		if num == -5313 {
			break
		}

		// Jika input adalah 0, hitung median
		if num == 0 {
```

```
                    if len(data) > 0 {
                          sort.Ints(data) // Urutkan data
                          median := calculateMedian(data)
                          fmt.Println(median)
                    }
              } else {
                    // Tambahkan angka ke dalam data
                    data = append(data, num)
              }
        }
}
```

Screenshoot Output

```
Masukkan deretan angka, pisahkan dengan spasi. Akhiri dengan -5313:
7 23 11 0 5 19 2 29 3 13 17 0 -5313
11
12
```

Deskripsi Program

Kode program di atas adalah sebuah aplikasi berbasis konsol yang ditulis dalam bahasa Go untuk membaca dan memproses deretan angka dari pengguna. Program dimulai dengan meminta pengguna untuk memasukkan serangkaian angka yang dipisahkan oleh spasi, dengan instruksi bahwa angka -5313 digunakan untuk mengakhiri input. Input ini diproses satu per satu, di mana angka-angka tersebut akan dimasukkan ke dalam sebuah slice (array dinamis) bernama data.

Jika pengguna memasukkan angka 0, program akan menghitung median dari elemen-elemen dalam slice data. Untuk menghitung median, elemen-elemen dalam slice terlebih dahulu diurutkan menggunakan fungsi sort.Ints, kemudian median dihitung menggunakan fungsi calculateMedian. Median ditentukan sebagai nilai tengah jika jumlah elemen ganjil, atau rata-rata dari dua nilai tengah jika jumlah elemen genap. Median yang dihitung kemudian ditampilkan ke layar.

Jika pengguna memasukkan angka selain 0 atau -5313, angka tersebut ditambahkan ke dalam slice data. Jika ada kesalahan dalam konversi input menjadi angka, program akan menghentikan eksekusi dan memberikan pesan kesalahan. Program berakhir ketika pengguna memasukkan -5313. Kode ini mengimplementasikan logika dasar untuk menangani masukan dinamis dan menghitung statistik sederhana (median) secara interaktif.

Unguided 3

```
package main

import (
    "bufio"
    "fmt"
    "os"
    "sort"
    "strconv"
    "strings"
)

// Konstanta dan tipe data
const nMax = 7919

type Buku struct {
    id, judul, penulis, penerbit string
    eksemplar, tahun, rating     int
}

type DaftarBuku []Buku

// Fungsi untuk membaca input dari pengguna
func bacaInput() string {
    reader := bufio.NewReader(os.Stdin)
    input, _ := reader.ReadString('\n')
    return strings.TrimSpace(input)
}

// Input data buku ke pustaka
func DaftarkanBuku(pustaka *DaftarBuku, n int) {
    for i := 0; i < n; i++ {
        fmt.Printf("Masukkan data buku ke-%d (format: id
judul penulis penerbit eksemplar tahun rating):\n", i+1)
        data := bacaInput()
        fields := strings.Fields(data)
        if len(fields) < 7 {
            fmt.Println("Data tidak lengkap. Ulangi!")
            i--
            continue
        }

        eksemplar, _ := strconv.Atoi(fields[4])
        tahun, _ := strconv.Atoi(fields[5])
        rating, _ := strconv.Atoi(fields[6])

        buku := Buku{
            id:       fields[0],
            judul:    fields[1],
            penulis:  fields[2],
            penerbit: fields[3],
            eksemplar: eksemplar,
```

```
            tahun:     tahun,
            rating:    rating,
        }
        *pustaka = append(*pustaka, buku)
    }
}

// Menampilkan buku terfavorit
func CetakTerfavorit(pustaka DaftarBuku, n int) {
    if n == 0 {
        fmt.Println("Tidak ada buku.")
        return
    }
    maxRating := -1
    var favorit Buku
    for _, buku := range pustaka {
        if buku.rating > maxRating {
            maxRating = buku.rating
            favorit = buku
        }
    }
    fmt.Printf("Buku Terfavorit:\nJudul: %s\nPenulis:
%s\nPenerbit: %s\nTahun: %d\n",
        favorit.judul, favorit.penulis,
favorit.penerbit, favorit.tahun)
}

// Mengurutkan pustaka berdasarkan rating secara menurun
func UrutBuku(pustaka *DaftarBuku, n int) {
    sort.Slice(*pustaka, func(i, j int) bool {
        return (*pustaka)[i].rating >
(*pustaka)[j].rating
    })
}

// Menampilkan 5 buku dengan rating tertinggi
func Cetak5Terbaru(pustaka DaftarBuku, n int) {
    fmt.Println("5 Buku dengan Rating Tertinggi:")
    for i := 0; i < n && i < 5; i++ {
        fmt.Printf("%d. %s (Rating: %d)\n", i+1,
pustaka[i].judul, pustaka[i].rating)
    }
}

// Mencari buku dengan rating tertentu menggunakan
pencarian biner
func CariBuku(pustaka DaftarBuku, n int, r int) {
    low, high := 0, n-1
    for low <= high {
        mid := (low + high) / 2
        if pustaka[mid].rating == r {
            buku := pustaka[mid]
```

```
            fmt.Printf("Buku Ditemukan:\nJudul:
%s\nPenulis: %s\nPenerbit: %s\nTahun: %d\nEksemplar:
%d\nRating: %d\n",
                buku.judul, buku.penulis, buku.penerbit,
buku.tahun, buku.eksemplar, buku.rating)
            return
        } else if pustaka[mid].rating < r {
            high = mid - 1
        } else {
            low = mid + 1
        }
    }
    fmt.Println("Tidak ada buku dengan rating seperti
itu.")
}

func main() {
    var n, targetRating int
    var pustaka DaftarBuku

    // Input jumlah buku
    fmt.Println("Masukkan jumlah buku:")
    n, _ = strconv.Atoi(bacaInput())

    // Daftarkan buku
    DaftarkanBuku(&pustaka, n)

    // Cetak buku terfavorit
    CetakTerfavorit(pustaka, n)

    // Urutkan pustaka berdasarkan rating
    UrutBuku(&pustaka, n)

    // Cetak 5 buku terbaru
    Cetak5Terbaru(pustaka, n)

    // Input rating yang dicari
    fmt.Println("Masukkan rating yang ingin dicari:")
    targetRating, _ = strconv.Atoi(bacaInput())

    // Cari buku berdasarkan rating
    CariBuku(pustaka, n, targetRating)
}
```

Screenshoot Output

```
Masukkan data buku ke-1 (format: id judul penulis penerbit eksemplar
    tahun rating):
1 dilan vin gramedia p 2020 5
Buku Terfavorit:
Judul: dilan
Penulis: vin
Penerbit: gramedia
Tahun: 2020
5 Buku dengan Rating Tertinggi:
1. dilan (Rating: 5)
Masukkan rating yang ingin dicari:
5
Buku Ditemukan:
Judul: dilan
Penulis: vin
Penerbit: gramedia
Tahun: 2020
Eksemplar: 0
```

Deskripsi Program

Program ini adalah aplikasi berbasis konsol dalam bahasa Go untuk mengelola data buku di perpustakaan. Pengguna diminta untuk memasukkan jumlah buku, kemudian menginput data setiap buku yang meliputi ID, judul, penulis, penerbit, jumlah eksemplar, tahun terbit, dan rating. Data ini disimpan dalam struktur slice bernama DaftarBuku. Program kemudian melakukan berbagai operasi seperti mencari buku dengan rating tertinggi (buku terfavorit), mengurutkan buku berdasarkan rating secara menurun, dan menampilkan hingga lima buku dengan rating tertinggi. Selain itu, program memungkinkan pengguna untuk mencari buku dengan rating tertentu menggunakan metode pencarian biner setelah data diurutkan. Program ini memanfaatkan fungsi modular untuk memastikan efisiensi, tetapi dapat ditingkatkan lebih lanjut dengan menambahkan validasi input dan penanganan kesalahan yang lebih robust.