**LAPORAN PRAKTIKUM  
ALGORITMA PEMROGRAMAN 2**

**MODUL XII**

**PENGURUTAN DATA**



Oleh:

RIZKULLOH ALPRIYANSAH

2311102142

IF-11-08

**PROGRAM STUDI S1 INFORMATIKA**

**FAKULTAS INFORMATIKA**

**UNIVERSITAS TELKOM PURWOKERTO**

**2024**

1. **DASAR TEORI**

Pengurutan data adalah proses menyusun elemen-elemen dalam suatu kumpulan data (array, slice, atau list) berdasarkan urutan tertentu, seperti secara **ascending (menaik)** atau **descending (menurun)**. Pengurutan ini bertujuan untuk mempermudah proses pencarian, pengolahan, dan analisis data.

Pengurutan dapat dibedakan menjadi dua jenis utama:

1. Pengurutan Internal: Dilakukan sepenuhnya di dalam memori utama (RAM). Cocok untuk dataset kecil hingga sedang.
2. Pengurutan Eksternal: Digunakan untuk dataset besar yang tidak dapat ditampung sepenuhnya di memori utama.

Beberapa algoritma pengurutan yang sering digunakan meliputi:

* Bubble Sort: Algoritma sederhana yang menukar elemen berurutan jika tidak sesuai dengan urutan yang diinginkan.
* Selection Sort: Memilih elemen terkecil (atau terbesar) dari data yang belum terurut, lalu meletakkannya di posisi yang sesuai.
* Insertion Sort: Menyisipkan elemen ke posisi yang benar dalam subset data yang sudah terurut.
* Merge Sort: Membagi data menjadi bagian-bagian kecil, mengurutkannya, lalu menggabungkannya kembali.
* Quick Sort: Memilih elemen pivot untuk membagi data ke dalam dua bagian dan mengurutkannya secara rekursif.
* Heap Sort: Menggunakan struktur data heap untuk menyusun data dalam urutan.

Implementasi Pengurutan di Go

Di Go, pengurutan data biasanya dilakukan menggunakan fungsi bawaan dari paket sort. Paket ini mendukung pengurutan untuk tipe data dasar (seperti int, float64, dan string) dan pengurutan khusus untuk tipe data yang lebih kompleks.

a. Pengurutan untuk Slice Tipe Data Dasar

Paket sort menyediakan fungsi seperti:

* sort.Ints(slice []int) untuk mengurutkan slice bertipe int.
* sort.Strings(slice []string) untuk mengurutkan slice bertipe string.
* sort.Float64s(slice []float64) untuk mengurutkan slice bertipe float64.

Kompleksitas Waktu

Kompleksitas waktu pengurutan bergantung pada algoritma yang digunakan:

* Bubble Sort: O(n²)
* Selection Sort: O(n²)
* Insertion Sort: O(n²)
* Merge Sort: O(n log n)
* Quick Sort: O(n log n) (rata-rata), O(n²) (kasus terburuk)
* Heap Sort: O(n log n)

Paket sort di Go menggunakan algoritma hybrid yang menggabungkan Quick Sort, Heap Sort, dan Insertion Sort untuk kinerja optimal.

Kesimpulan

Pengurutan data adalah aspek penting dalam pemrograman yang berguna untuk meningkatkan efisiensi pencarian dan manipulasi data. Go menyediakan fitur bawaan yang efisien melalui paket sort, mendukung pengurutan data dasar maupun tipe data kompleks.

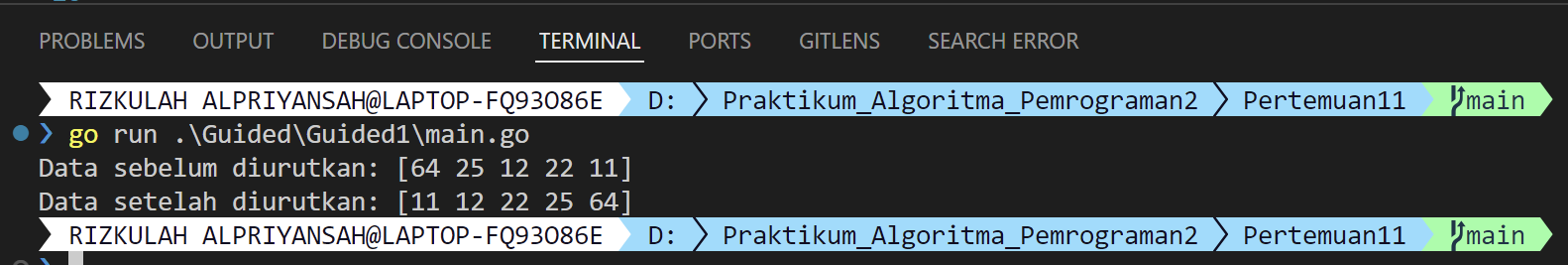
1. **GUIDED**

**Guided1**

**Source Code**

|  |
| --- |
| package main  import "fmt"  // Fungsi untuk melakukan Selection Sort  func selectionSort(arr []int) {      n := len(arr)      for i := 0; i < n-1; i++ { // Iterasi dari indeks pertama sampai kedua terakhir          idxMin := i // Anggap elemen pada indeks i adalah yang terkecil            // Cari elemen terkecil di sisa array          for j := i + 1; j < n; j++ {              if arr[j] < arr[idxMin] {                  idxMin = j // Update indeks nilai terkecil              }          }            // Tukar elemen terkecil dengan elemen di posisi i          if idxMin != i {              temp := arr[i]              arr[i] = arr[idxMin]              arr[idxMin] = temp          }      }  }    func main() {      // Data yang akan diurutkan      data := []int{64, 25, 12, 22, 11}        fmt.Println("Data sebelum diurutkan:", data)        // Panggil fungsi Selection Sort      selectionSort(data)        fmt.Println("Data setelah diurutkan:", data)  } |

**ScreenShot Output**

****

**Deskripsi**

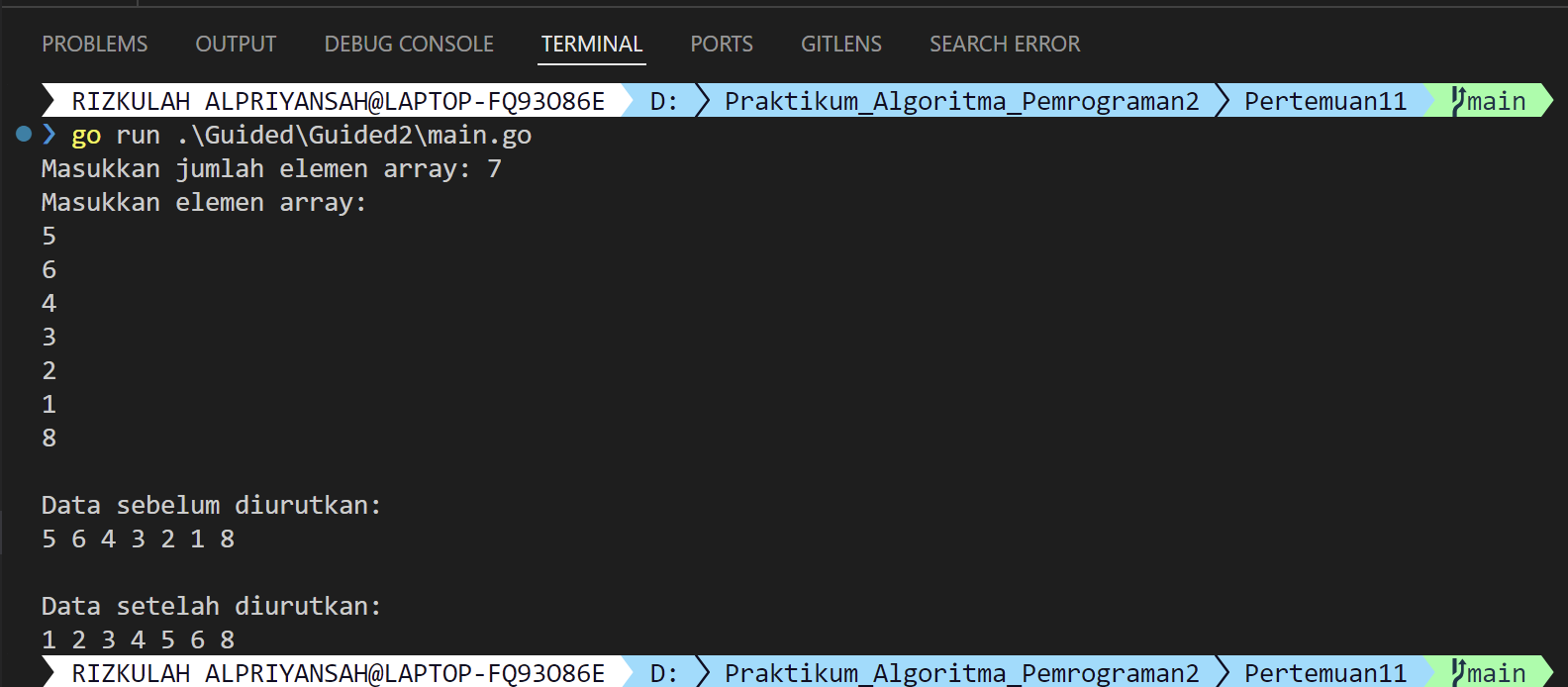
Program ini mengimplementasikan algoritma *Selection Sort* dalam bahasa Go untuk mengurutkan array integer secara *ascending*. Fungsi selectionSort menerima array sebagai parameter, kemudian iterasi dilakukan untuk menemukan elemen terkecil di sisa array dan menukarnya dengan elemen pada indeks saat ini. Jika elemen terkecil ditemukan di indeks lain, proses *swap* dilakukan. Pada fungsi main, data array diberikan, kemudian fungsi selectionSort dipanggil untuk mengurutkan array tersebut, dan hasilnya ditampilkan sebelum dan setelah diurutkan.

**Guided2**

**Source Code**

|  |
| --- |
| package main  import "fmt"    type arrInt [4321]int    func selectionSort1(T \*arrInt, n int) {      var t, i, j, idx\_min int        i = 1      for i <= n-1 {          idx\_min = i - 1          j = i          for j < n {              if T[idx\_min] > T[j] {                  idx\_min = j              }              j = j + 1          }          t = T[idx\_min]          T[idx\_min] = T[i-1]          T[i-1] = t          i = i + 1      }  }    func main() {      var data arrInt      var n int        fmt.Print("Masukkan jumlah elemen array: ")      fmt.Scan(&n)        fmt.Println("Masukkan elemen array:")      for i := 0; i < n; i++ {          fmt.Scan(&data[i])      }        fmt.Println("\nData sebelum diurutkan:")      for i := 0; i < n; i++ {          fmt.Printf("%d ", data[i])  }  selectionSort1(&data, n)  fmt.Println("\n\nData setelah diurutkan:")  for i := 0; i < n; i++ {  }  } |

**ScreenShot Output**



**Deskripsi**

Program di atas mengimplementasikan algoritma *Selection Sort* dalam bahasa Go untuk mengurutkan array yang elemen dan ukurannya dimasukkan oleh pengguna. Tipe data arrInt digunakan untuk mendeklarasikan array dengan kapasitas maksimum 4321 elemen. Fungsi selectionSort1 menerima array dan jumlah elemennya sebagai parameter, menggunakan pointer untuk memungkinkan modifikasi langsung pada array asli. Algoritma *Selection Sort* bekerja dengan mencari elemen terkecil di sisa array dan menukarnya dengan elemen pada posisi saat ini. Fungsi main meminta pengguna untuk memasukkan jumlah elemen array (n) dan nilainya, kemudian menampilkan array sebelum dan sesudah diurutkan. Seluruh proses pengurutan dilakukan secara langsung pada array menggunakan fungsi selectionSort1.

**Guided3**

**Source Code**

|  |
| --- |
| package main  import "fmt"    type mahasiswa struct {      nama, nim, kelas, jurusan string      ipk                       float64  }    type arrMhs [2023]mahasiswa    func selectionSort2(T \*arrMhs, n int) {        var i, j, idx\_min int      var t mahasiswa        i = 1      for i <= n-1 {          idx\_min = i - 1          j = i          for j < n {              if T[idx\_min].ipk > T[j].ipk {                  idx\_min = j              }              j = j + 1          }          t = T[idx\_min]          T[idx\_min] = T[i-1]          T[i-1] = t          i = i + 1      }  }    func main() {      var data arrMhs      var n int        fmt.Print("Masukkan jumlah mahasiswa: ")      fmt.Scan(&n)        fmt.Println("Masukkan data mahasiswa (Nama, NIM, Kelas, Jurusan, IPK):")      for i := 0; i < n; i++ {          fmt.Printf("Data mahasiswa ke-%d:\n", i+1)          fmt.Print("Nama: ")          fmt.Scan(&data[i].nama)          fmt.Print("NIM: ")          fmt.Scan(&data[i].nim)          fmt.Print("Kelas: ")          fmt.Scan(&data[i].kelas)          fmt.Print("Jurusan: ")          fmt.Scan(&data[i].jurusan)          fmt.Print("IPK: ")          fmt.Scan(&data[i].ipk)      }        fmt.Println("\nData mahasiswa sebelum diurutkan:")      for i := 0; i < n; i++ {          fmt.Printf("%s (%s) - %s - %s - IPK: %.2f\n",  data[i].nama, data[i].nim, data[i].kelas, data[i].jurusan,  data[i].ipk)      }      selectionSort2(&data, n)        fmt.Println("\nData mahasiswa setelah diurutkan berdasarkan IPK (terkecil-terbesar):")      for i := 0; i < n; i++ {          fmt.Printf("%s (%s) - %s - %s - IPK: %.2f\n",  data[i].nama, data[i].nim, data[i].kelas, data[i].jurusan,  data[i].ipk)      }  } |

**ScreenShot Output**

A screenshot of a computer

Description automatically generated

**Deskripsi**

Program ini mengimplementasikan algoritma *Selection Sort* dalam bahasa Go untuk mengurutkan data mahasiswa berdasarkan IPK (Indeks Prestasi Kumulatif) dari yang terkecil ke yang terbesar. Tipe data mahasiswa merepresentasikan data mahasiswa yang terdiri dari atribut nama, nim, kelas, jurusan, dan ipk. Tipe arrMhs digunakan untuk mendeklarasikan array mahasiswa dengan kapasitas maksimum 2023 elemen. Fungsi selectionSort2 mengurutkan array mahasiswa menggunakan algoritma *Selection Sort*, di mana elemen dengan IPK terkecil dipindahkan ke posisi awal secara iteratif. Pada fungsi main, pengguna diminta memasukkan jumlah mahasiswa dan datanya, yang kemudian ditampilkan sebelum dan sesudah pengurutan. Program ini menampilkan daftar mahasiswa yang telah diurutkan berdasarkan IPK dalam format yang terstruktur.

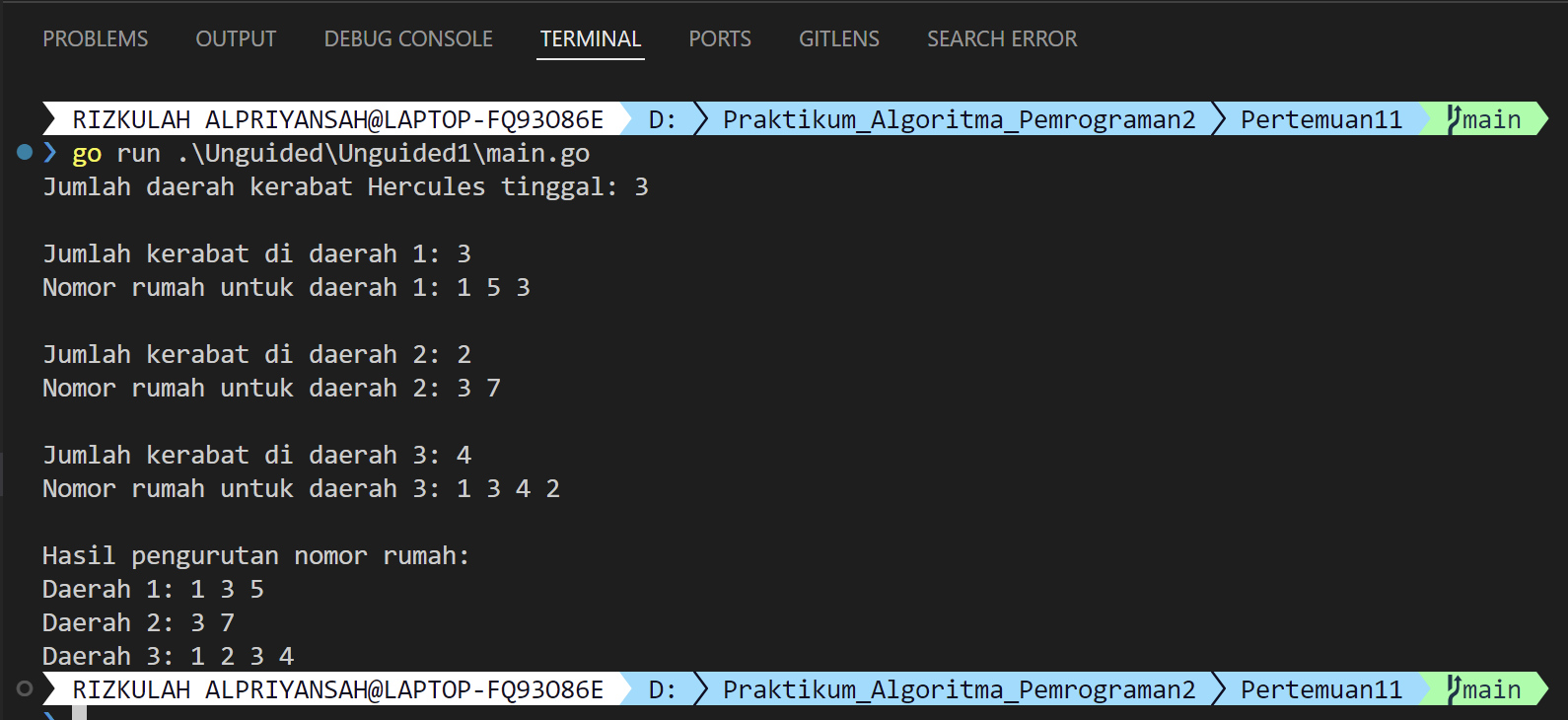
1. **UNGUIDED**

**Unguided1**

**Source Code**

|  |
| --- |
| package main  import (      "fmt"  )  func selectionSort\_142(arr\_142 []int) {      n\_142 := len(arr\_142)      for i\_142 := 0; i\_142 < n\_142-1; i\_142++ {          minIdx\_142 := i\_142          for j\_142 := i\_142 + 1; j\_142 < n\_142; j\_142++ {              if arr\_142[j\_142] < arr\_142[minIdx\_142] {                  minIdx\_142 = j\_142              }          }          arr\_142[i\_142], arr\_142[minIdx\_142] = arr\_142[minIdx\_142], arr\_142[i\_142]      }  }  func main() {      var n\_142 int      fmt.Print("Jumlah daerah kerabat Hercules tinggal: ")      fmt.Scan(&n\_142)      if n\_142 <= 0 || n\_142 > 1000 {          return      }      daerah\_142 := make([][]int, n\_142)      for i\_142 := 0; i\_142 < n\_142; i\_142++ {          var m\_142 int          fmt.Printf("\nJumlah kerabat di daerah %d: ", i\_142+1)          fmt.Scan(&m\_142)          if m\_142 <= 0 || m\_142 > 1000000 {              return          }          daerah\_142[i\_142] = make([]int, m\_142)          fmt.Printf("Nomor rumah untuk daerah %d: ", i\_142+1)          for j\_142 := 0; j\_142 < m\_142; j\_142++ {              fmt.Scan(&daerah\_142[i\_142][j\_142])          }          selectionSort\_142(daerah\_142[i\_142])      }      fmt.Println("\nHasil pengurutan nomor rumah:")      for i\_142 := 0; i\_142 < n\_142; i\_142++ {          fmt.Printf("Daerah %d: ", i\_142+1)          for j\_142 := 0; j\_142 < len(daerah\_142[i\_142]); j\_142++ {              fmt.Printf("%d ", daerah\_142[i\_142][j\_142])          }          fmt.Println()      }  } |

**Screenshot**



**Deskripsi**

Program di atas merupakan implementasi algoritma *Selection Sort* dalam bahasa Go yang digunakan untuk mengurutkan nomor rumah kerabat Hercules di berbagai daerah tempat tinggalnya. Pengguna diminta untuk memasukkan jumlah daerah (n\_142) dan jumlah kerabat di masing-masing daerah (m\_142), diikuti dengan daftar nomor rumah setiap kerabat. Data untuk setiap daerah disimpan dalam array dua dimensi daerah\_142. Fungsi selectionSort\_142 bertugas mengurutkan nomor rumah secara *ascending* menggunakan algoritma *Selection Sort*. Setelah semua data diurutkan, program menampilkan nomor rumah kerabat di setiap daerah dalam urutan yang telah disusun. Program juga memiliki validasi untuk memastikan jumlah daerah tidak melebihi 1000 dan jumlah kerabat tidak lebih dari 1 juta per daerah.

**Unguided2**

**Source Code**

|  |
| --- |
| package main  import (      "fmt"  )  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 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 main() {      var n int      fmt.Print("Masukkan jumlah daerah (n): ")      fmt.Scan(&n)      results := make([][]int, n)      for i := 0; i < n; i++ {          var m int          fmt.Printf("Masukkan jumlah rumah kerabat di daerah %d: ", i+1)          fmt.Scan(&m)          rumahKerabat := make([]int, m)          fmt.Printf("Masukkan nomor rumah kerabat di daerah %d: ", i+1)          for j := 0; j < m; j++ {              fmt.Scan(&rumahKerabat[j])          }          var ganjil, genap []int          for \_, rumah := range rumahKerabat {              if rumah%2 == 1 {                  ganjil = append(ganjil, rumah)              } else {                  genap = append(genap, rumah)              }          }          fmt.Println("Ganjil sebelum sorting:", ganjil)          fmt.Println("Genap sebelum sorting:", genap)          selectionSortDesc(ganjil)          selectionSortAsc(genap)          fmt.Println("Ganjil setelah sorting:", ganjil)          fmt.Println("Genap setelah sorting:", genap)          results[i] = append(ganjil, genap...)      }      fmt.Println("\nHasil:")      for i, result := range results {          fmt.Printf("Daerah %d: ", i+1)          for \_, rumah := range result {              fmt.Printf("%d ", rumah)          }          fmt.Println()      }  } |

**Screenshot**

A screenshot of a computer

Description automatically generated

**Deskripsi**

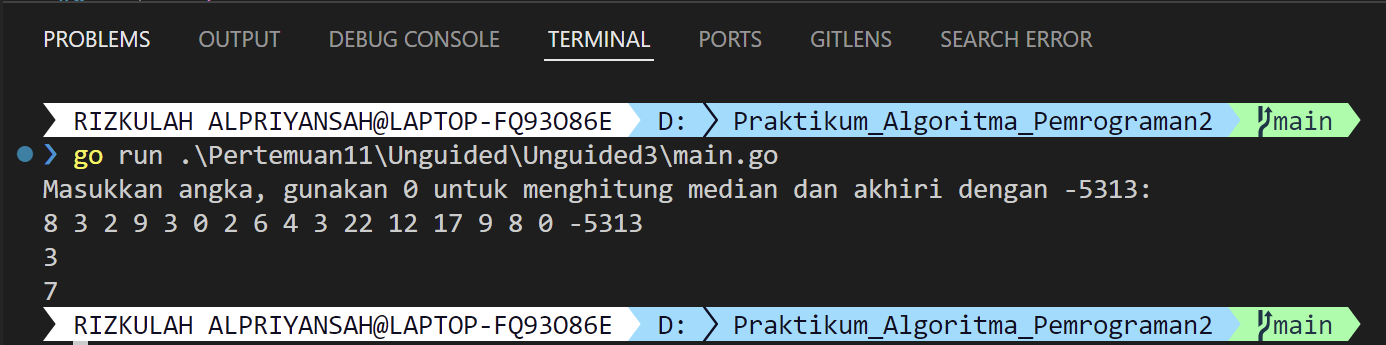
Program di atas mengimplementasikan algoritma selection sort untuk mengurutkan nomor rumah kerabat di beberapa daerah. Input terdiri dari jumlah daerah dan nomor rumah di tiap daerah. Setiap nomor rumah dikelompokkan menjadi ganjil dan genap: nomor ganjil diurutkan secara menurun (descending) dan nomor genap diurutkan secara menaik (ascending) menggunakan fungsi selectionSortDesc dan selectionSortAsc. Hasil pengelompokan dan pengurutan digabungkan kembali untuk setiap daerah dan ditampilkan di akhir. Program ini membantu mengorganisasi data nomor rumah secara terstruktur sesuai kategori dan urutan yang diinginkan.

**Unguided3**

**Source Code**

|  |
| --- |
| package main  import (      "bufio"      "fmt"      "os"      "sort"      "strconv"      "strings"  )  func findMedian\_142(numbers\_142 []int) int {      sort.Ints(numbers\_142)      n\_142 := len(numbers\_142)      if n\_142%2 == 1 {          return numbers\_142[n\_142/2]      }      return (numbers\_142[n\_142/2-1] + numbers\_142[n\_142/2]) / 2  }  func main() {      reader\_142 := bufio.NewReader(os.Stdin)      fmt.Println("Masukkan angka, gunakan 0 untuk menghitung median dan akhiri dengan -5313:")      input\_142, \_ := reader\_142.ReadString('\n')      input\_142 = strings.TrimSpace(input\_142)      data\_142 := strings.Split(input\_142, " ")      var numbers\_142 []int      for \_, value\_142 := range data\_142 {          num\_142, err\_142 := strconv.Atoi(value\_142)          if err\_142 != nil {              fmt.Println("Input tidak valid, masukkan hanya angka.")              return          }          if num\_142 == -5313 {              break          } else if num\_142 == 0 {              if len(numbers\_142) > 0 {                  median\_142 := findMedian\_142(numbers\_142)                  fmt.Println(median\_142)              }          } else {              numbers\_142 = append(numbers\_142, num\_142)          }      }  } |

**Screenshot**



**Deskripsi**

Program di atas menerima input angka dari pengguna yang dimasukkan dalam satu baris, dipisahkan oleh spasi. Program akan mengolah angka-angka yang dimasukkan dan menghitung median ketika pengguna memasukkan angka 0. Proses dimulai dengan membaca input menggunakan bufio.Reader, lalu angka-angka diproses dan disimpan dalam slice numbers\_142. Ketika angka 0 dimasukkan, program akan menghitung median dari angka yang sudah dimasukkan hingga saat itu menggunakan fungsi findMedian\_142. Fungsi findMedian\_142 mengurutkan angka terlebih dahulu dan kemudian mencari median berdasarkan jumlah elemen (genap atau ganjil). Proses akan berhenti jika pengguna memasukkan angka -5313. Program ini juga memvalidasi input untuk memastikan hanya angka yang dimasukkan.

1. **DAFTAR PUSTAKA**

1) Asisten praktikum, Akmelia Zahara dan Kyla Azzahra Kinan “Modul XII PENGURUTAN DATA” Learning Management System, 2024