LAPORAN PRAKTIKUM
ALGORITMA DAN PEMROGRAMAN 2

MODUL XI

PENCARIAN NILAI ACAK PADA HIMPUNAN DATA

Oleh:

RINDI DELA NUR SAFITRI

2311102332

IF-11-07

S1 TEKNIK INFORMATIKA

UNIVERSITAS TELKOM PURWOKERTO

2024

I. DASAR TEORI

11.1 Sequential Search

Pencarian secara sekuensial ini adalah pencarian yang dilakukan dari data pertama, kedua hingga terakhir secara satu persatu dan berurutan. Ciri khas dari pencarian ini adalah proses pencarian akan berhenti ketika data yang dicari ditemukan, walaupun masih terdapat data yang belum dicek nilainya. Algoritma ini dikenal dengan nama Sequential Search, karena prosesnya melakukan pengecekan setiap elemen array secara satu persatu dan sekuensial dari data pertama hingga ditemukan atau data terakhir.

1) Asumsi terdapat suatu array of integer T dengan indeks dari 0 hingga N-1, dan suatu nilai yang dicari pada array T, yaitu X.
2) Status pencarian digunakan untuk menandakan data yang dicari ditemukan atau tidak, misalnya found dengan tipe boolean.
3) Pencarian dilakukan dari T[0] sampai ke T[N-1], setiap kali perbandingan dengan X, update nilai found.
4) Perulangan harus dihentikan apabila status pencarian found bernilai true (data ditemukan) atau T[N-1] telah dicek.

11.2 Binary Search

Ide algoritma adalah: (dengan asumsi data terurut dari kecil membesar (ascending), dan data dengan indeks kecil ada di "kiri" dan indeks besar ada di "kanan")

1) Ambil salah satu data dalam rentang data yang ada, algoritma di bawah menggunakan rentang dari kiri $\boldsymbol{kr}$ s.d. kanan $\boldsymbol{kn}$. Untuk kemudahan dan alasan lainnya, biasanya yang diambil adalah paling tengah dalam rentang tersebut.
2) Jika data terambil tersebut terlalu kecil, maka ubah/geser rentang data ke sebelah kanan posisi data tersebut. Ini karena jika data terambil terlalu kecil, maka semua data sebalah kirinya juga akan terlalu kecil dari yang ingin dicari.
3) Begitu juga sebaliknya jika data terambil terlalu besar.

11.3 Pencarian pada Array Bertipe Data Struct

Algoritma pencarian pada array bertipe data struct tidak berbeda jauh dengan tipe data dasar, kita hanya perlu menambahkan field kategori dari pencarian yang akan dilakukan. Khusus untuk algoritma binary search, maka keterurutan array harus sama dengan field kategori dari pencariannya. Misalnya apabila pencarian berdasarkan nim mahasiswa, maka algoritma binary search hanya akan bisa digunakan apabila array terurut berdasarkan nim. Algoritma binary search tidak akan berjalan apabila array terurut berdasarkan nim mahasiswa, tetapi pencarian yang dilakukan berdasarkan filed nama atau kategori yang lain selain nim. Solusinya adalah array harus diurutkan berdasarkan nim terlebih dahulu, atau gunakan algoritma sequential search.

II. GUIDED

Guided 1

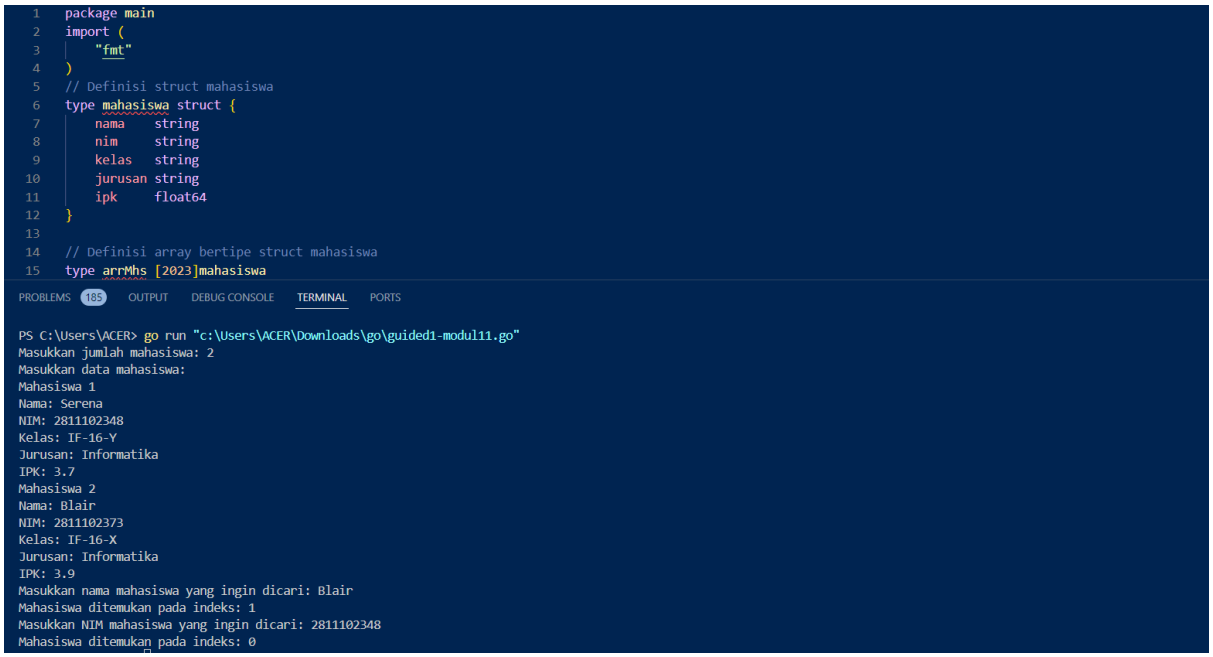
Source Code

```
package main
import (
    "fmt"
)
// Definisi struct mahasiswa
type mahasiswa struct {
    nama    string
    nim     string
    kelas   string
    jurusan string
    ipk     float64
}

// Definisi array bertipe struct mahasiswa
type arrMhs [2023]mahasiswa

// Sequential Search berdasarkan nama
func SeqSearch_3(T arrMhs, n int, X string) int {
    /* Mengembalikan indeks mahasiswa dengan nama X,
       atau -1 apabila tidak ditemukan pada array T
       yang berisi n data mahasiswa */
    var found int = -1
    var j int = 0
    for j < n && found == -1 {
        if T[j].nama == X {
            found = j
        }
        j = j + 1
    }
    return found
}

// Binary Search berdasarkan nim
func BinarySearch_3(T arrMhs, n int, X string) int {
    /* Mengembalikan indeks mahasiswa dengan nim X,
       atau -1 apabila tidak ditemukan pada array T
       yang berisi n data mahasiswa dan terurut membesar berdasarkan nim
*/
    var found int = -1
    var med int
    var kr int = 0
    var kn int = n - 1

    for kr <= kn && found == -1 {
        med = (kr + kn) / 2
```

```
        if X < T[med].nim {
            kn = med - 1
        } else if X > T[med].nim {
            kr = med + 1
        } else {
            found = med
        }
    }
    return found
}

// Fungsi utama untuk demonstrasi
func main() {
    var data arrMhs
    var n int

    // Input jumlah mahasiswa
    fmt.Print("Masukkan jumlah mahasiswa: ")
    fmt.Scanln(&n)

    // Input data mahasiswa
    fmt.Println("Masukkan data mahasiswa:")
    for i := 0; i < n; i++ {
        fmt.Printf("Mahasiswa %d\n", i+1)
        fmt.Print("Nama: ")
        fmt.Scanln(&data[i].nama)
        fmt.Print("NIM: ")
        fmt.Scanln(&data[i].nim)
        fmt.Print("Kelas: ")
        fmt.Scanln(&data[i].kelas)
        fmt.Print("Jurusan: ")
        fmt.Scanln(&data[i].jurusan)
        fmt.Print("IPK: ")
        fmt.Scanln(&data[i].ipk)
    }

    // Pencarian berdasarkan nama
    var cariNama string
    fmt.Print("Masukkan nama mahasiswa yang ingin dicari: ")
    fmt.Scanln(&cariNama)
    posNama := SeqSearch_3(data, n, cariNama)
    if posNama == -1 {
        fmt.Println("Mahasiswa dengan nama tersebut tidak ditemukan.")
    } else {
        fmt.Printf("Mahasiswa ditemukan pada indeks: %d\n", posNama)
    }
```

```
    // Pencarian berdasarkan nim
    var cariNIM string
    fmt.Print("Masukkan NIM mahasiswa yang ingin dicari: ")
    fmt.Scanln(&cariNIM)
    posNIM := BinarySearch_3(data, n, cariNIM)
    if posNIM == -1 {
        fmt.Println("Mahasiswa dengan NIM tersebut tidak ditemukan.")
    } else {
        fmt.Printf("Mahasiswa ditemukan pada indeks: %d\n", posNIM)
    }
}
```

Screenshot Program



Penjelasan

Program di atas untuk mencari data mahasiswa dalam suatu himpunan menggunakan dua metode pencarian: Sequential Search dan Binary Search. Program ini pertama kali meminta pengguna untuk memasukkan jumlah mahasiswa, lalu menginputkan detail informasi mahasiswa seperti nama, NIM, kelas, jurusan, dan IPK ke dalam sebuah array statis bertipe `mahasiswa`. Setelah semua data mahasiswa diinput, program menyediakan dua metode pencarian, yaitu pencarian berdasarkan nama menggunakan Sequential Search dan pencarian berdasarkan NIM menggunakan Binary Search.

Sequential Search digunakan untuk mencari mahasiswa berdasarkan nama dengan cara memeriksa elemen array satu per satu secara linear hingga menemukan data yang sesuai atau mencapai akhir array. Di sisi lain, Binary Search digunakan untuk pencarian berdasarkan NIM. Binary Search bekerja lebih efisien dengan asumsi bahwa data telah terurut berdasarkan NIM, membagi himpunan data menjadi dua secara iteratif hingga menemukan data yang dicari. Metode ini lebih cepat dibandingkan Sequential Search, terutama untuk data besar, namun memerlukan data terurut sebagai prasyarat. Kedua metode akan memberikan indeks mahasiswa jika ditemukan, atau -1 jika tidak ditemukan dalam himpunan.

Guided 2

Source Code

```
package main

import "fmt"
// Define arrInt as an array of integers with a fixed size
type arrInt [4321]int
// Binary search function for descending sorted array
func BinarySearch_2(T arrInt, n int, X int) int {
    /* Mengembalikan indeks dari X apabila X ditemukan di dalam array T
       yang berisi n buah bilangan bulat terurut secara
descending/menciut,
       atau -1 apabila X tidak ditemukan */
    var found int = -1
    var med int
    var kr int = 0
    var kn int = n - 1

    // Iterasi mencari elemen dengan binary search
    for kr <= kn && found == -1 {
        med = (kr + kn) / 2
        if X > T[med] { // Karena descending, jika X lebih besar,
bergerak ke kiri
            kn = med - 1
        } else if X < T[med] { // Jika X lebih kecil, bergerak ke kanan
            kr = med + 1
        } else { // Jika ditemukan
            found = med
        }
    }
    return found
}

func main() {
    // Contoh penggunaan
    var data arrInt
    var n int
    // Input jumlah elemen
    fmt.Print("Masukkan jumlah elemen dalam array: ")
    fmt.Scanln(&n)
    // Input elemen array (harus terurut secara descending)
    fmt.Println("Masukkan elemen array (harus terurut menurun):")
    for i := 0; i < n; i++ {
        fmt.Printf("Elemen %d: ", i+1)
        fmt.Scanln(&data[i])
    }
```

```
    // Input elemen yang dicari
    var search int
    fmt.Print("Masukkan elemen yang ingin dicari: ")
    fmt.Scanln(&search)

    // Panggil fungsi binary search
    result := BinarySearch_2(data, n, search)

    // Cetak hasil
    if result == -1 {
        fmt.Println("Elemen tidak ditemukan dalam array.")
    } else {
        fmt.Printf("Elemen ditemukan pada indeks: %d\n", result)
    }
}
```

Screenshot Program

Penjelasan

Program di atas merupakan implementasi Binary Search untuk mencari elemen dalam array yang telah terurut secara menurun (descending). Array yang digunakan memiliki tipe `arrInt`, sebuah array statis dengan kapasitas maksimal 4321 elemen. Fungsi `BinarySearch_2` mencari elemen tertentu, `X`, dalam array. Logikanya disesuaikan dengan kondisi array terurut secara menurun: jika nilai tengah lebih kecil dari `X`, maka pencarian diarahkan ke bagian kiri array, dan jika nilai tengah lebih besar dari `X`, pencarian diarahkan ke bagian kanan. Jika elemen ditemukan, fungsi mengembalikan indeksnya; jika tidak, hasilnya adalah `-1`.

Dalam fungsi `main`, pengguna diminta untuk memasukkan jumlah elemen array (`n`) dan elemen-elemen tersebut dalam urutan menurun. Setelah itu, pengguna dapat memasukkan nilai

yang ingin dicari. Fungsi `BinarySearch_2` dipanggil untuk menemukan indeks elemen tersebut. Program ini memberikan efisiensi lebih baik dibandingkan pencarian linear pada array besar, karena Binary Search membagi rentang pencarian menjadi dua setiap iterasi. Namun, program mengharuskan array sudah terurut menurun sebagai prasyarat agar pencarian bekerja dengan benar.

Guided 3

Source Code

```
package main
import "fmt"
// Define arrStr as an array of strings with a fixed size
type arrStr [1234]string
// Sequential search function
func SeqSearch_1(T arrStr, n int, X string) int {
    /* Mengembalikan indeks dari X apabila X ditemukan di dalam array T
       yang berisi n buah teks, atau -1 apabila X tidak ditemukan */
    var found int = -1
    var j int = 0

    // Iterasi mencari elemen yang cocok
    for j < n && found == -1 {
        if T[j] == X {
            found = j
        }
        j = j + 1
    }
    return found
}
func main() {
    // Contoh penggunaan
    var data arrStr
    var n int

    // Input jumlah elemen
    fmt.Print("Masukkan jumlah elemen dalam array: ")
    fmt.Scanln(&n)

    // Input elemen array
    fmt.Println("Masukkan elemen array:")
    for i := 0; i < n; i++ {
        fmt.Printf("Elemen %d: ", i+1)
        fmt.Scanln(&data[i])
    }
    // Input elemen yang dicari
    var search string
    fmt.Print("Masukkan elemen yang ingin dicari: ")
    fmt.Scanln(&search)
```

```
    // Panggil fungsi sequential search
    result := SeqSearch_1(data, n, search)
    // Cetak hasil
    if result == -1 {
        fmt.Println("Elemen tidak ditemukan dalam array.")
    } else {
        fmt.Printf("Elemen ditemukan pada indeks: %d\n", result)
    }
}
```

Screenshot Program

Penjelasan

Program di atas merupakan implementasi Sequential Search untuk mencari elemen berupa string dalam sebuah array. Array didefinisikan dengan tipe `arrStr`, yaitu array statis berkapasitas maksimal 1234 elemen. Fungsi `SeqSearch_1` melakukan pencarian linear untuk menemukan indeks elemen tertentu, `X`, dalam array. Pencarian dilakukan dengan memeriksa setiap elemen dari awal hingga akhir. Jika elemen yang dicari ditemukan, indeksnya dikembalikan; jika tidak, fungsi mengembalikan nilai `-1`.

Pada fungsi `main`, pengguna diminta untuk memasukkan jumlah elemen array (`n`) dan memasukkan elemen-elemen berupa string ke dalam array. Setelah array terisi, pengguna diminta memasukkan string yang ingin dicari. Fungsi `SeqSearch_1` kemudian dipanggil untuk mencari elemen tersebut. Hasil pencarian akan berupa indeks elemen jika ditemukan, atau pesan bahwa elemen tidak ditemukan. Program ini cocok digunakan pada data berukuran kecil hingga sedang, karena pencarian linear kurang efisien dibandingkan metode seperti Binary Search pada data yang besar atau terurut.

III. UNGUIDED

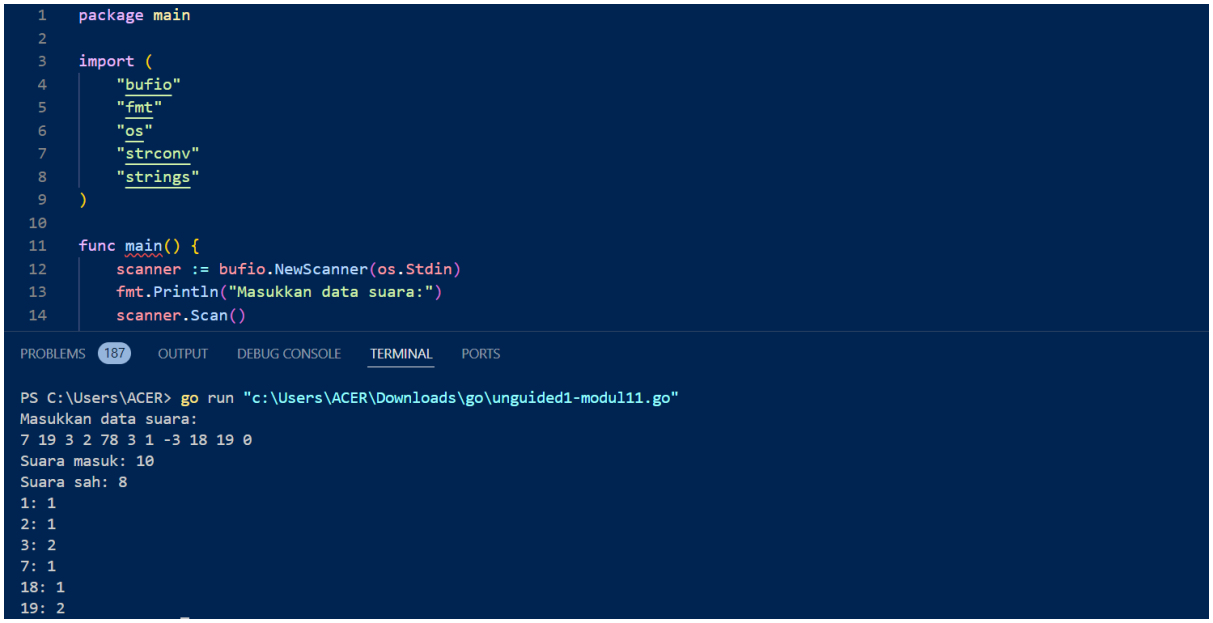
Unguided 1

Source Code

```
package main

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

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Println("Masukkan data suara (pisahkan dengan spasi, akhiri
dengan 0):")
    scanner.Scan()
    input := scanner.Text()

    data := strings.Split(input, " ")

    var suaraMasuk, suaraSah int
    var count [20]int

    for _, v := range data {

        num, err := strconv.Atoi(v)
        if err != nil {
            fmt.Println("Input tidak valid!")
            return
        }

        if num == 0 {
            break
        }

        suaraMasuk++
        if num >= 1 && num <= 20 {
            suaraSah++

            count[num-1]++
        }
    }

    fmt.Printf("Suara masuk: %d\n", suaraMasuk)
    fmt.Printf("Suara sah: %d\n", suaraSah)
```

```
    for i := 0; i < 20; i++ {
        if count[i] > 0 {
            fmt.Printf("%d: %d\n", i+1, count[i])
        }
    }
}
```

Screenshot Program



Penjelasan

Program di atas merupakan sebuah aplikasi yang digunakan untuk menghitung suara yang diterima oleh 20 kandidat dalam suatu pemilu. Pengguna diminta untuk memasukkan data suara dalam bentuk angka, dipisahkan oleh spasi, dan diakhiri dengan angka 0. Program kemudian memproses input tersebut dengan cara melakukan pencarian secara sekuensial (sequential search). Setiap angka suara yang dimasukkan akan dikonversi menjadi integer dan diperiksa apakah valid (dalam rentang 1 hingga 20). Jika valid, program menghitung jumlah suara sah dan menambahkannya ke dalam array `count` yang menyimpan jumlah suara untuk setiap kandidat.

Dalam pendekatan sequential search, program memeriksa setiap angka yang dimasukkan satu per satu untuk memastikan apakah angka tersebut valid dan dalam rentang yang telah ditentukan. Setiap suara yang valid akan diperiksa dan dihitung secara bertahap untuk menentukan jumlah suara sah yang diterima oleh masing-masing kandidat. Pendekatan ini disebut pencarian sekuensial karena program tidak menggunakan teknik pencarian lain yang lebih efisien, tetapi secara langsung memeriksa setiap elemen input dengan urutan yang tetap, dari awal hingga akhir.

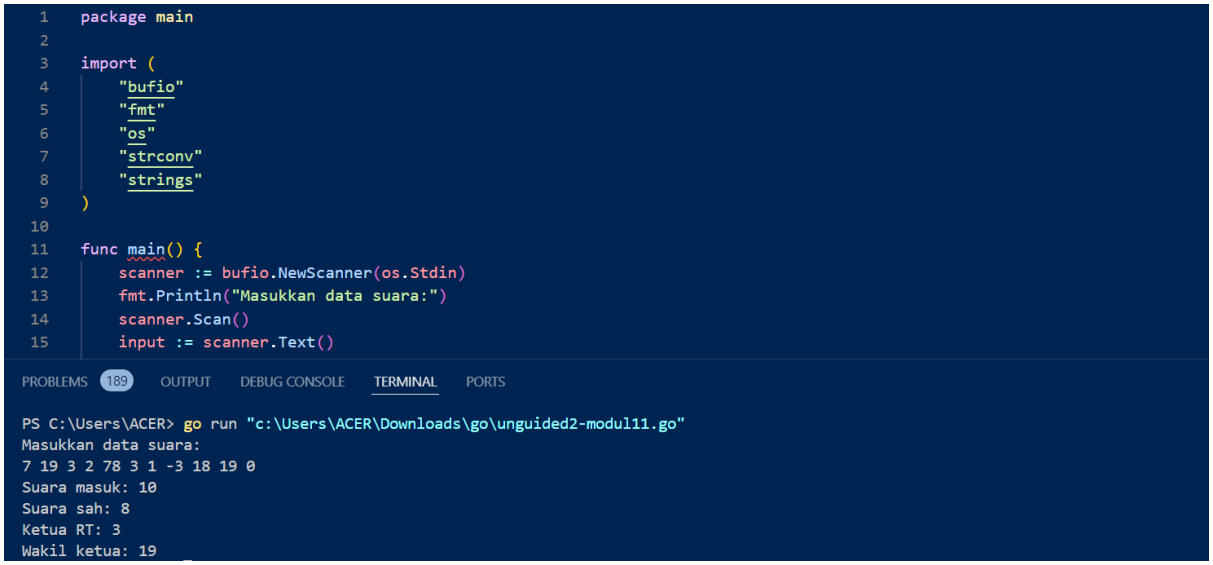
Unguided 2

Source Code

```
package main

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

func main() {
    scanner := bufio.NewScanner(os.Stdin)
    fmt.Println("Masukkan data suara:")
    scanner.Scan()
    input := scanner.Text()

    data := strings.Split(input, " ")

    var suaraMasuk, suaraSah int
    var count [20]int

    for _, v := range data {

        num, err := strconv.Atoi(v)
        if err != nil {

            continue
        }

        if num == 0 {
            break
        }

        suaraMasuk++

        if num >= 1 && num <= 20 {
            suaraSah++
            count[num-1]++
        }
    }

    fmt.Printf("Suara masuk: %d\n", suaraMasuk)
    fmt.Printf("Suara sah: %d\n", suaraSah)

    var ketua, wakil int
    var maxSuara, secondMaxSuara int
```

```
    for i := 0; i < 20; i++ {
        if count[i] > maxSuara {
            maxSuara = count[i]
            ketua = i + 1
        }
    }

    for i := 0; i < 20; i++ {

        if count[i] > secondMaxSuara && i+1 != ketua {
            secondMaxSuara = count[i]
            wakil = i + 1
        }
    }

    fmt.Printf("Ketua RT: %d\n", ketua)
    fmt.Printf("Wakil ketua: %d\n", wakil)
}
```

Screenshot Program



Penjelasan

Program di atas merupakan aplikasi yang menghitung suara yang diterima oleh 20 kandidat dalam sebuah pemilu dan menentukan ketua serta wakil ketua berdasarkan jumlah suara terbanyak. Input suara diberikan oleh pengguna, kemudian program memproses data untuk menghitung suara masuk, suara sah, dan jumlah suara untuk setiap kandidat. Program ini menggunakan pendekatan pencarian sekuensial (sequential search) untuk menemukan kandidat dengan suara terbanyak dan kedua terbanyak.

Dalam pencarian sekuensial, program memeriksa setiap elemen array `count` satu per satu untuk menemukan kandidat dengan jumlah suara terbanyak (ketua), lalu melakukan hal yang sama untuk mencari suara terbanyak kedua (wakil ketua), dengan pengecualian terhadap ketua yang sudah terpilih. Proses ini dilakukan tanpa menggunakan metode pencarian yang lebih

kompleks, sehingga seluruh elemen dicek secara berurutan. Ini adalah penerapan dari pencarian sekuensial, di mana setiap elemen array diproses dalam urutan yang tetap.

Unguided 3

Source Code

```
package main

import "fmt"

const NMAX = 1000000
var data [NMAX]int

func main() {
    var n, k int
    fmt.Print("Masukkan n dan k: ")
    fmt.Scanf("%d %d", &n, &k)

    isiArray(n)
    posisiIndex := posisi(n, k)

    if posisiIndex == -1 {
        fmt.Println("Keluaran: TIDAK ADA")
    } else {
        fmt.Printf("Keluaran: %d\n", posisiIndex)
    }
}

func isiArray(n int) {
    fmt.Print("Masukkan data: ")
    for i := 0; i < n; i++ {
        fmt.Scanf("%d", &data[i])
    }
}

func posisi(n, k int) int {
    kiri, kanan := 0, n-1

    for kiri <= kanan {
        tengah := (kiri + kanan) / 2

        if data[tengah] == k {
            return tengah
        } else if data[tengah] < k {
            kiri = tengah + 1
        } else {
            kanan = tengah - 1
        }
    }

    return -1
}
```

Screenshot Program

```go
package main

import "fmt"

const NMAX = 1000000
var data [NMAX]int

func main() {
    var n, k int
    fmt.Print("Masukkan n dan k: ")
    fmt.Scanf("%d %d", &n, &k)

    isiArray(n)
    posisiIndex := posisi(n, k)
```

```
PROBLEMS 196  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\ACER> go run "c:\Users\ACER\Downloads\go\unguided3-modul11.go"
Masukkan n dan k: 12 535
Masukkan data: 1 3 8 16 32 123 323 323 534 543 823 999
Keluaran: TIDAK ADA
```

Penjelasan

Program di atas mencari posisi suatu elemen dalam array menggunakan algoritma binary search. Pertama, program meminta pengguna untuk memasukkan dua angka, yaitu ukuran array `n` dan nilai yang ingin dicari `k`. Setelah itu, program meminta input data array sebanyak `n` elemen. Fungsi `isiArray(n)` digunakan untuk mengisi array dengan data yang diberikan oleh pengguna. Kemudian, fungsi `posisi(n, k)` digunakan untuk mencari posisi elemen `k` dalam array yang sudah diinputkan menggunakan metode binary search. Algoritma ini bekerja dengan membagi array menjadi dua bagian secara berulang untuk mencari elemen yang dicari.

Dalam fungsi `posisi(n, k)`, dua variabel `kiri` dan `kanan` digunakan untuk menyimpan batas pencarian, dan pencarian dilakukan dengan menghitung indeks tengah. Jika nilai di posisi tengah sama dengan `k`, maka indeks tengah tersebut dikembalikan. Jika nilai tengah lebih kecil dari `k`, maka pencarian dilanjutkan pada setengah bagian kanan, dan jika lebih besar, pencarian dilanjutkan pada setengah bagian kiri. Jika elemen tidak ditemukan, fungsi ini akan mengembalikan `-1`, menandakan bahwa elemen yang dicari tidak ada dalam array. Program kemudian menampilkan hasil pencarian, baik posisi elemen atau pesan bahwa elemen tidak ditemukan.