LAPORAN TUGAS BESAR ANALISIS KOMPLEKSITAS ALGORITMA

Nama kelompok :

- **Christ Daniel Santoso (2311102305)**
- **Naswa Malika Putri (2311102232)**

IF-11-02

Dosen :

Maie Istighosah, S.Kom., M.Kom

S1 Teknik Informatika

Telkom University Purwokerto

2024

LAPORAN ANALISIS PROGRAM
CARDS SORTING WITH SELECTION
AND BINARY INSERTION SORT

A. Study Case Permasalahan

Dalam aplikasi *games* kartu remi, pemain sering kali menerima kartu acak yang perlu diurutkan dari nilai terkecil ke terbesar untuk mempermudah strategi permainan. Pengurutan manual dapat memakan waktu dan mengurangi pengalaman bermain, sehingga diperlukan solusi otomatis yang efisien. Program ini menggunakan algoritma selection sort dan binary insertion sort untuk mengurutkan kartu secara otomatis sesuai kebutuhan lalu mengevaluasi mana algoritma yang paling cocok digunakan berdasarkan banyaknya kartu acak yang diberikan.

B. Deskripsi Program

Program ini dirancang untuk mengurutkan kartu remi berdasarkan nilai dan jenisnya secara efisien menggunakan beberapa algoritma sorting, yaitu Selection Sort (iteratif dan rekursif) serta Binary Insertion Sort (iteratif dan rekursif). Nilai kartu diurutkan dari yang terkecil ke yang terbesar, jika terdapat nilai yang sama, kartu akan diurutkan berdasarkan prioritas: ♠ (Sekop), ♥ (Hati), ♣ (Keriting), dan ♦ (Wajik).

Program dimulai dengan membuat kartu remi acak sebanyak inputan yang ditentukan. Pengguna dapat memilih untuk menampilkan kartu sebelum dan sesudah diurutkan. Proses pengurutan dilakukan dengan empat metode: Selection Sort iteratif, Selection Sort rekursif, Binary Insertion Sort iteratif, dan Binary Insertion Sort rekursif. Program juga mengukur waktu eksekusi masing-masing metode guna membandingkan performa dan evaluasi. Dengan fleksibilitas dan efisiensi yang ditawarkan, program ini sangat cocok digunakan dalam simulasi *games* kartu remi atau aplikasi yang memerlukan pengurutan data serupa.

A. Source Code Program

```
package main

import (
    "bufio"
    "fmt"
    "math/rand"
    "os"
    "strings"
    "time"
)

type Kartu struct {
    Angka int
    Jenis string
}

var urutanJenis = map[string]int{
    "♠": 1,
    "♥": 2,
    "♣": 3,
    "♦": 4,
}

func formatAngka(angka int) string {
    switch angka {
    case 1:
        return "AS"
    case 11:
        return "Jack"
    case 12:
        return "Queen"
    case 13:
        return "King"
    default:
        return fmt.Sprintf("%d", angka)
    }
}

func bandingkanKartu(a, b Kartu) bool {
    if a.Angka != b.Angka {
        return a.Angka < b.Angka
    }
    return urutanJenis[a.Jenis] > urutanJenis[b.Jenis]
```

```
}

func selectionSortIteratif(kartu []Kartu) []Kartu {
    n := len(kartu)
    for i := 0; i < n-1; i++ {
        indeksMin := i
        for j := i + 1; j < n; j++ {
            if bandingkanKartu(kartu[j], kartu[indeksMin]) {
                indeksMin = j
            }
        }
        kartu[i], kartu[indeksMin] = kartu[indeksMin], kartu[i]
    }
    return kartu
}

func selectionSortRekursif(kartu []Kartu, mulai int) []Kartu {
    n := len(kartu)
    if mulai >= n-1 {
        return kartu
    }
    indeksMin := mulai
    for i := mulai + 1; i < n; i++ {
        if bandingkanKartu(kartu[i], kartu[indeksMin]) {
            indeksMin = i
        }
    }
    kartu[mulai], kartu[indeksMin] = kartu[indeksMin], kartu[mulai]
    return selectionSortRekursif(kartu, mulai+1)
}

func binaryInsertionSortIteratif(kartu []Kartu) []Kartu {
    n := len(kartu)
    for i := 1; i < n; i++ {
        kunci := kartu[i]
        low, high := 0, i-1
        for low <= high {
            mid := (low + high) / 2
            if bandingkanKartu(kunci, kartu[mid]) {
                high = mid - 1
            } else {
                low = mid + 1
            }
        }
```

```
        for j := i; j > low; j-- {
            kartu[j] = kartu[j-1]
        }
        kartu[low] = kunci
    }
    return kartu
}

func binaryInsertionSortRekursif(kartu []Kartu, n int) []Kartu {
    if n <= 1 {
        return kartu
    }
    kartu = binaryInsertionSortRekursif(kartu, n-1)
    kunci := kartu[n-1]
    low, high := 0, n-2
    for low <= high {
        mid := (low + high) / 2
        if bandingkanKartu(kunci, kartu[mid]) {
            high = mid - 1
        } else {
            low = mid + 1
        }
    }
    for j := n - 1; j > low; j-- {
        kartu[j] = kartu[j-1]
    }
    kartu[low] = kunci
    return kartu
}

func tampilkanKartu(kartu []Kartu) {
    for i, k := range kartu {
        fmt.Printf("[%-7s %s] ", formatAngka(k.Angka), k.Jenis)
        if (i+1)%7 == 0 {
            fmt.Println()
        }
    }
    if len(kartu)%7 != 0 {
        fmt.Println()
    }
}

func main() {
    kartu := []Kartu{}
```

```
jumlahKartu := 250

reader := bufio.NewReader(os.Stdin)
fmt.Print("Apakah anda ingin menampilkan data kartu? (y/n): ")
tampilkanData, _ := reader.ReadString('\n')
tampilkanData = strings.TrimSpace(tampilkanData)

rand.Seed(time.Now().UnixNano())

for i := 0; i < jumlahKartu; i++ {
    kartu = append(kartu, Kartu{
        Angka: i%13 + 1,
        Jenis: []string{"♠", "♥", "♣", "♦"}[i%4],
    })
}

if strings.ToLower(tampilkanData) == "y" {
    fmt.Println("\nData Kartu Sebelum Diurutkan:")
    tampilkanKartu(kartu)
} else {
    fmt.Println("\nUser memilih untuk tidak menampilkan data kartu")
}

mulai := time.Now()
hasilSelectionIteratif := selectionSortIteratif(append([]Kartu{}, kartu...))
waktu := time.Since(mulai).Seconds()

if strings.ToLower(tampilkanData) == "y" {
    fmt.Println("\nHasil Kartu Setelah Diurutkan:")
    tampilkanKartu(hasilSelectionIteratif)
}

fmt.Printf("\nWaktu Running Time Selection Sort (Iteratif): %.6f detik\n", waktu)

mulai = time.Now()
_ = binaryInsertionSortIteratif(append([]Kartu{}, kartu...))
waktu = time.Since(mulai).Seconds()
fmt.Printf("Waktu Running Time Binary Insertion Sort (Iteratif): %.6f detik\n", waktu)

mulai = time.Now()
_ = selectionSortRekursif(append([]Kartu{}, kartu...), 0)
waktu = time.Since(mulai).Seconds()
fmt.Printf("Waktu Running Time Selection Sort (Rekursif): %.6f detik\n", waktu)
```

```
    mulai = time.Now()
    _ = binaryInsertionSortRekursif(append([]Kartu{}, kartu...), len(kartu))
    waktu = time.Since(mulai).Seconds()
    fmt.Printf("Waktu Running Time Binary Insertion Sort (Rekursif): %.6f detik\n", waktu)
}
```

Deskripsi

Program diatas bertujuan untuk meng-*generate* data kartu remi secara acak sebanyak variabel jumlahKartu. Lalu menggunakan empat (4) algoritma, seluruh kartu acak akan diurutkan. Keempat algoritma tersebut adalah :

- Selection Sort (Pendekatan Iteratif)
- Selection Sort (Pendekatan Rekursif)
- Binary Insertion Sort (Pendekatan Iteratif)
- Binary Insertion Sort (Pendekatan Rekursif)

Program juga akan mengukur *running time* dari keempat algoritma tersebut guna perbandingan performa.

Fungsi Utama

```
func formatAngka(angka int) string {
        switch angka {
        case 1:
                return "AS"
        case 11:
                return "Jack"
        case 12:
                return "Queen"
        case 13:
                return "King"
        default:
                return fmt.Sprintf("%d", angka)
        }
}
```

Func ***formatAngka*** : Mengubah nilai angka kartu menjadi representasi visual kartu remi (contoh: 1 menjadi "AS", 11 menjadi "Jack").

```
func bandingkanKartu(a, b Kartu) bool {
```

```
        if a.Angka != b.Angka {
                return a.Angka < b.Angka
        }
        return urutanJenis[a.Jenis] > urutanJenis[b.Jenis]
}
```

Func ***bandingkanKartu*** : Membandingkan dua kartu berdasarkan nilai (Angka) dan jenis (♠, ♥, ♣, ♦) untuk menentukan urutan.

```
func selectionSortIteratif(kartu []Kartu) []Kartu {
        n := len(kartu)
        for i := 0; i < n-1; i++ {
                indeksMin := i
                for j := i + 1; j < n; j++ {
                        if bandingkanKartu(kartu[j], kartu[indeksMin]) {
                                indeksMin = j
                        }
                }
                kartu[i], kartu[indeksMin] = kartu[indeksMin], kartu[i]
        }
        return kartu
}
```

Func ***selectionSortIteratif*** : Mengurutkan kartu secara iteratif menggunakan algoritma Selection Sort.

```
func selectionSortRekursif(kartu []Kartu, mulai int) []Kartu {
        n := len(kartu)
        if mulai >= n-1 {
                return kartu
        }
        indeksMin := mulai
        for i := mulai + 1; i < n; i++ {
                if bandingkanKartu(kartu[i], kartu[indeksMin]) {
                        indeksMin = i
                }
        }
        kartu[mulai], kartu[indeksMin] = kartu[indeksMin], kartu[mulai]
        return selectionSortRekursif(kartu, mulai+1)
}
```

Func ***selectionSortRekursif*** : Mengurutkan kartu secara rekursif menggunakan algoritma Selection Sort.

```
func binaryInsertionSortIteratif(kartu []Kartu) []Kartu {
	n := len(kartu)
	for i := 1; i < n; i++ {
		kunci := kartu[i]
		low, high := 0, i-1
		for low <= high {
			mid := (low + high) / 2
			if bandingkanKartu(kunci, kartu[mid]) {
				high = mid - 1
			} else {
				low = mid + 1
			}
		}
		for j := i; j > low; j-- {
			kartu[j] = kartu[j-1]
		}
		kartu[low] = kunci
	}
	return kartu
}
```

Func ***binaryInsertionSortIteratif*** : Mengurutkan kartu secara iteratif menggunakan algoritma Binary Insertion Sort.

```
func binaryInsertionSortRekursif(kartu []Kartu, n int) []Kartu {
	if n <= 1 {
		return kartu
	}
	kartu = binaryInsertionSortRekursif(kartu, n-1)
	kunci := kartu[n-1]
	low, high := 0, n-2
	for low <= high {
		mid := (low + high) / 2
		if bandingkanKartu(kunci, kartu[mid]) {
			high = mid - 1
		} else {
			low = mid + 1
		}
```

```
        }
        for j := n - 1; j > low; j-- {
                kartu[j] = kartu[j-1]
        }
        kartu[low] = kunci
        return kartu
}
```

Func ***binaryInsertionSortRekursif*** : Mengurutkan kartu secara rekusif menggunakan algoritma Binary Insertion Sort.

```
func tampilkanKartu(kartu []Kartu) {
        for i, k := range kartu {
                fmt.Printf(" [%-7s %-1s", formatAngka(k.Angka), k.Jenis)
                fmt.Print("] ")
                if (i+1)%7 == 0 {
                        fmt.Println()
                }
        }
        if len(kartu)%7 != 0 {
                fmt.Println()
        }
}
```

Func ***tampilkanKartu*** : Menampilkan daftar kartu kedalam format yang rapi, dengan maksimal 7 kartu per baris.

B. Hasil *Running Time* (Waktu Eksekusi) Program

Algoritam selection Sort (Iteratif)

Data Masukan 0 : 0.000000
Data Masukan 200 : 0.000000
Data Masukan 400 : 0.000525
Data Masukan 800 : 0.001704
Data Masukan 1600 : 0.006937

Algoritma Selection Sort (Rekursif)

Data Masukan 0 : 0.000000
Data Masukan 200 : 0.000000
Data Masukan 400 : 0.000511
Data Masukan 800 : 0.003703
Data Masukan 1600 : 0.008500

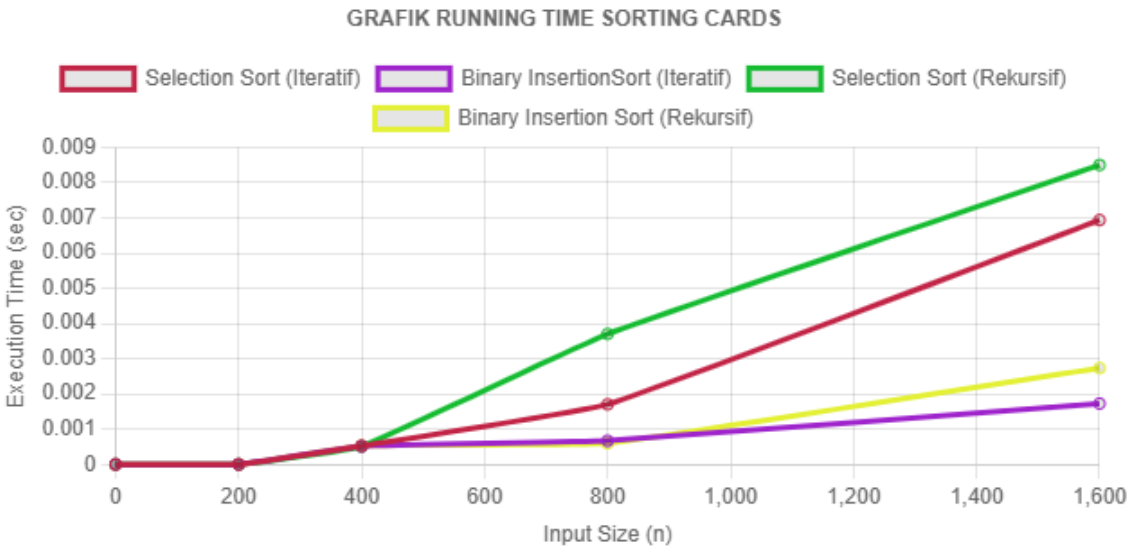
Algoritma Binary Insertion Sort (Iteratif)

Data Masukan 0 : 0.000000
Data Masukan 200 : 0.000000
Data Masukan 400 : 0.000517
Data Masukan 800 : 0.000672
Data Masukan 1600 : 0.001727

Algoritma Binary Insertion Sort (Rekursif)

Data Masukan 0 : 0.000000
Data Masukan 200 : 0.000000
Data Masukan 400 : 0.000526
Data Masukan 800 : 0.000600
Data Masukan 1600 : 0.002735

C. Grafik Running Time



D. Analisis *Running Time* dan Kompleksitas Waktu Tiap Fungsi

1. Fungsi ***formatAngka***
   a. Operasi Dasar

      Switch-case untuk memeriksa nilai angka → O(1).

      Mengembalikan string menggunakan fmt.Sprintf → O(1).

   b. Frekuensi

      Hanya satu pemeriksaan nilai angka dan satu pengembalian nilai string.

   c. Kompleksitas Waktu

      O(1).

2. Fungsi ***bandingkanKartu***
   a. Operasi Dasar

      Perbandingan nilai angka a.Angka != b.Angka → O(1).

      Perbandingan jenis kartu urutanJenis[a.Jenis] > urutanJenis[b.Jenis] → O(1).

   b. Frekuensi

      Maksimal dua operasi dasar untuk setiap pasangan kartu yang dibandingkan.

   c. Kompleksitas Waktu

      O(1).

3. Fungsi ***selectionSortIteratif***
    a. Operasi Dasar

        Perbandingan kartu dengan bandingkanKartu → O(1).

        Pertukaran posisi kartu kartu[i], kartu[indeksMin] = kartu[indeksMin], kartu[i] → O(1).
    b. Frekuensi

        Untuk setiap elemen n, dilakukan.$n-1, n-2$, ..., hingga 1 perbandingan.

        Total Operasi dasar → $\frac{n(n-1)}{2}$
    c. Kompleksitas Waktu

        O(n^2).

4. Fungsi ***selectionSortRekursif***
    a. Operasi Dasar

        Perbandingan kartu dengan bandingkanKartu → O(1).

        Pertukaran posisi kartu → O(1).

        Pemanggilan rekursif → O(n).
    b. Frekuensi

        Untuk setiap elemen n, dilakukan.$n-1, n-2$, ..., hingga 1 perbandingan.

        Total Operasi dasar → $\frac{n(n-1)}{2}$

        Rekursi dilakukan hingga $n-1$ kali.
    c. Kompleksitas Waktu

        O(n^2).

5. Fungsi ***binaryInsertionSortIteratif***
    a. Operasi Dasar

        Pencarian posisi dengan binary search → O(log n).

        Pergeseran elemen → O(n).
    b. Frekuensi

Untuk setiap elemen n. dilakukan pencarian posisi dan pergeseran elemen.

Total operasi dasar → $n \cdot (logn + n)$.

c. Kompleksitas Waktu

O(n^2).

6. Fungsi ***binaryInsertionSortRekursif***

a. Operasi Dasar

Pencarian posisi dengan binary search → O(log n).

Pergeseran elemen → O(n).

Pemanggilan rekursif → O(n).

b. Frekuensi

Untuk setiap elemen n. dilakukan pencarian posisi dan pergeseran elemen.

Total operasi dasar → $n \cdot (logn + n)$.

c. Kompleksitas Waktu

O(n^2).

7. Fungsi ***tampilkanKartu***

a. Operasi Dasar

Iterasi melalui elemen kartu untuk pencetakan → O(1).

Pemformatan string dengan fmt.Printf → O(1).

b. Frekuensi

Dilakukan sebanyak n kali untuk setiap elemen kartu.

c. Kompleksitas Waktu

O(n).

E. Analisis Total Program

Semua fungsi sorting dan fungsi bantu dipanggil secara independen dalam program utama. Berdasarkan analisis kompleksitas waktu masing-masing fungsi, berikut adalah total kompleksitas program:

1. Fungsi ***selectionSortIteratif***

Kompleksitas waktu: $O(n^2)$

2. Fungsi ***binaryInsertionSortIteratif***

   Kompleksitas waktu: $O(n^2)$

3. Fungsi ***selectionSortRekursif***

   Kompleksitas waktu: $O(n^2)$

4. Fungsi ***binaryInsertionSortRekursif***

   Kompleksitas waktu: $O(n^2)$

5. Fungsi ***tampilkanKaru***

   Kompleksitas waktu: $O(n)$

Total Kompleksitas Waktu

Karena semua fungsi sorting dipanggil secara independen, total kompleksitas waktu adalah***:***

$$T(n) = O(n^2) + O(n^2) + O(n^2) + O(n^2) + O(n) + O(n)$$

$$T(n) = 4 \cdot O(n^2) + 2 \cdot O(n)$$

Untuk input besar, komponen $O(n^2)$ mendominasi, sehingga :

$$T(n) = O(n^2)$$

Jika jumlah data n meningkat secara linear, waktu eksekusi dari program akan meningkat secara kuadrat. Hal ini disebabkan oleh kompleksitas kuadratik dari algoritma sorting yang digunakan. Namun, untuk fungsi rekursif, ada tambahan overhead memori untuk setiap pemanggilan rekursif, yang dapat memperburuk performa dibandingkan versi iteratif.

F. Analisis Perbedaan Algoritma Sorting

Berdasarkan pengujian waktu eksekusi dan menganalisis running time serta kompleksitas waktu dari **kedua algoritma sorting**, hasil pengujian tersebut dapat dirangkum sebagai berikut:

Kriteria	**Selection Sort**	**Binary Insertion Sort**
Efisiensi	Tidak terlalu efisien untuk dataset besar karena kompleksitas $O(n^2)$	Lebih efisien dibanding Selection Sort dengan kompleksitas mendekati $O\ (n \log n)$
Kompleksitas	Sederhana, cocok untuk pembelajaran dasar algoritma	Lebih kompleks tetapi menghasilkan performa lebih baik pada dataset besar
Implementasi Iteratif	Mudah diimplementasikan dan digunakan	Agak rumit karena memerlukan operasi pencarian biner dalam array
Implementasi Rekursif	Lebih kompleks dan meningkatkan overhead memori	Masih kompleks, tetapi cocok untuk dataset kecil atau rekursi alami
Waktu Eksekusi	Cenderung lebih lambat	Lebih cepat untuk dataset besar, terutama pada implementasi iteratif
Grafik Iteratif	Linear meningkat stabil	Linear meningkat perlahan
Grafik rekursif	Linear meningkat curam	Linear meningkat cukup stabil namun tetap ada overhead stack

Berdasarkan perbandingan algoritma yang ditampilkan pada tabel dan analisis grafik:

- Binary Insertion Sort Iteratif adalah pilihan terbaik untuk efisiensi waktu, terutama pada dataset besar. Kompleksitasnya mendekati $O(n \log n)$ sehingga lebih optimal dibandingkan Selection Sort.
- Selection Sort Iteratif adalah pilihan yang lebih sederhana untuk implementasi, tetapi kurang efisien untuk dataset besar karena kompleksitas waktu $O(n^2)$
- Algoritma rekursif, baik Selection Sort maupun Binary Insertion Sort, memiliki kelemahan berupa overhead stack yang menyebabkan peningkatan waktu eksekusi.

Untuk studi kasus ini, Binary Insertion Sort Iteratif adalah pendekatan paling efisien dan praktis. Namun, jika kebutuhan algoritma lebih sederhana dan dataset kecil, Selection Sort Iteratif dapat digunakan.

G. Analisis Perbedaan Algoritma Perulangan

Berdasarkan pengujian waktu eksekusi dan menganalisis running time serta kompleksitas waktu dari kedua program, kita dapat hasil dari pengujian tersebut, yaitu:

Kriteria	**Iteratif**	**Rekursif**
Efisiensi	Lebih cepat untuk dataset yang besar	Sedikit lebih lambat karena terdapat overhead stack
Implementasi	Mudah dipahami dan digunakan	Lebih kompleks dan cocok untuk dataset kecil dan struktur rekursif alami
Waktu Eksekusi	Lebih cepat	Sedikit lebih cepat
Grafik	Linear stabil	Linear meningkat stabil

Penjelasan Analisis

1. Selection Sort (Iteratif vs Rekursif)

   Iteratif memiliki performa lebih stabil pada dataset besar karena tidak memiliki overhead pemanggilan fungsi.

   Rekursif menunjukkan waktu eksekusi yang lebih tinggi karena adanya tambahan overhead stack pada setiap pemanggilan fungsi.
2. Binary Insertion Sort (Iteratif vs Rekursif)

Iteratif menghasilkan waktu eksekusi yang lebih rendah pada dataset besar, karena prosesnya langsung tanpa melibatkan pemanggilan fungsi berulang.

Rekursif terlihat lebih efisien pada dataset kecil, tetapi performanya menurun signifikan ketika ukuran dataset bertambah.

3. Grafik Running Time

   Grafik iteratif (baik untuk selection sort maupun binary insertion sort) cenderung linear dan lebih stabil meskipun ukuran dataset bertambah.

   Grafik rekursif menunjukkan kenaikan yang lebih curam, terutama untuk dataset besar, karena beban pemanggilan fungsi yang berulang.

Berdasarkan perbandingan algoritma yang ditampilkan pada tabel serta studi kasus yang telah dianalisis, algoritma iteratif menjadi pilihan yang lebih sesuai. Hal ini dikarenakan kebutuhan program masih tergolong sederhana dan belum memiliki tingkat kompleksitas yang tinggi. Penggunaan algoritma iteratif memungkinkan efisiensi waktu eksekusi dan penghematan memori yang lebih baik. Sebaliknya, algoritma rekursif cenderung menimbulkan overhead memori yang lebih besar karena adanya proses pemanggilan fungsi secara berulang. Oleh karena itu, untuk studi kasus ini, algoritma iteratif dinilai sebagai pendekatan yang lebih efisien dan praktis.