TUGAS BESAR ANALISIS KOMPLEKSITAS ALGORITMA

ANALISIS KOMPLEKSITAS WAKTU ALGORITMA PENCARIAN SEKUENSIAL DAN BINER DALAM MENCARI TRANSAKSI PENGELUARAN

Disusun Oleh:

1. Michelina Patricia S. (103012300223)
2. Diana Nur Aisyah (103012300049)

:

S1 Informatika

Telkom University

A. Deskripsi studi kasus permasalahan

Studi kasus ini membahas penggunaan dua algoritma pencarian, yaitu pencarian sekuensial dan pencarian biner, untuk mencari suatu transaksi pengeluaran pada catatan transaksi yang banyak (lebih dari 300 transaksi). Tujuan dari proyek ini adalah untuk menganalisis kompleksitas waktu dari kedua algoritma dengan berbagai ukuran input dan kondisi data (terurut dan tidak terurut). Melalui pendekatan ini, dapat dianalisis kapan masing-masing algoritma lebih efisien dalam konteks mencari transaksi dalam sebuah database besar.

B. Deskripsi dua algoritma yang dipilih untuk menyelesaikan permasalahan

1. **Pencarian Sekuensial (Sequential Search)**
   - Algoritma ini melakukan pencarian elemen dengan memeriksa setiap elemen dalam daftar satu per satu dari awal hingga akhir.
   - Kompleksitas waktu: O(n) dalam kasus terburuk, di mana n adalah jumlah elemen dalam daftar.
2. **Pencarian Biner (Binary Search)**
   - Algoritma ini mencari elemen dalam daftar yang sudah terurut dengan membagi daftar menjadi dua bagian dan memeriksa apakah elemen tengah adalah elemen yang dicari. Jika tidak, algoritma menentukan bagian mana yang akan dicari selanjutnya, dan proses ini diulangi.
   - Kompleksitas waktu: O(log n) dalam kasus terburuk, di mana n adalah jumlah elemen dalam daftar.

C. Grafik perbandingan running time

Kode

TugasBesar.go × | Release Notes: 1.96.0 | pseudocode

TugasBesar.go > ...

```go
package main

import (
    "fmt"
    "math/rand"
    "sort"
    "time"
)

func generateTransactions(num int) []int {
    rand.Seed(time.Now().UnixNano())
    transactions := make([]int, num)
    for i := range transactions {
        transactions[i] = rand.Intn(1000)
    }
    return transactions
}

func sequentialSearch(transactions []int, target int) (bool, time.Duration) {
    start := time.Now()
    for _, transaction := range transactions {
        if transaction == target {
            return true, time.Since(start)
        }
    }
    return false, time.Since(start)
}

func binarySearch(transactions []int, target int) (bool, time.Duration) {
    start := time.Now()
    left, right := 0, len(transactions)-1
    for left <= right {
        mid := (left + right) / 2
        if transactions[mid] == target {
            return true, time.Since(start)
        } else if transactions[mid] < target {
            left = mid + 1
        } else {
            right = mid - 1
        }
    }
    return false, time.Since(start)
}

func main() {
    numTransactions := 300
    transactions := generateTransactions(numTransactions)
    target := transactions[rand.Intn(numTransactions)]

    found, duration := sequentialSearch(transactions, target)
    fmt.Printf("Sequential Search - Found: %v, Duration: %.9f seconds\n", found, duration.Seconds())

    sort.Ints(transactions)
    found, duration = binarySearch(transactions, target)
    fmt.Printf("Binary Search - Found: %v, Duration: %.9f seconds\n", found, duration.Seconds())
}
```

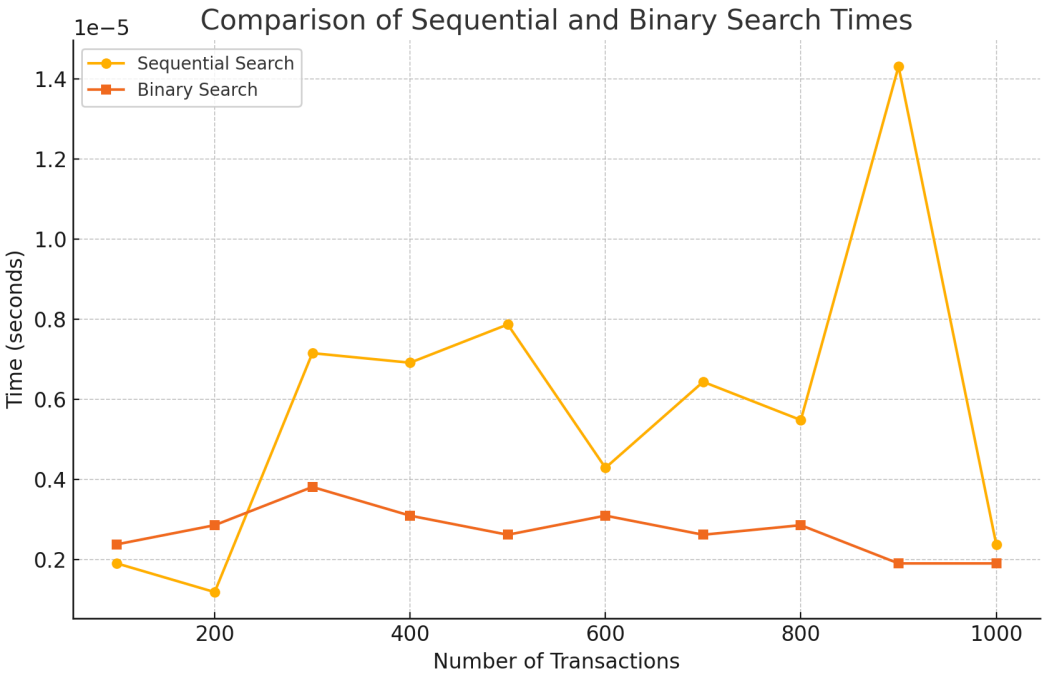
Hasil Kode

```
michelina@Michelinas-MacBook-Pro go_lang % go run TugasBesar.go
Sequential Search - Found: true, Duration: 0.000000292 seconds
Binary Search - Found: true, Duration: 0.000000125 seconds
michelina@Michelinas-MacBook-Pro go_lang %
```

Pseudocode

```pseudocode
SEQUENTIAL_SEARCH(transactions, target)
    start_time = CURRENT_TIME()

    FOR each transaction IN transactions
        IF transaction == target THEN
            return TRUE, CURRENT_TIME() - start_time
        END IF
    END FOR

    return FALSE, CURRENT_TIME() - start_time
END FUNCTION

---

BINARY_SEARCH(transactions, target)
    start_time = CURRENT_TIME()
    left = 0
    right = LENGTH(transactions) - 1

    WHILE left <= right DO
        mid = (left + right) // 2

        IF transactions[mid] == target THEN
            return TRUE, CURRENT_TIME() - start_time
        ELSE IF transactions[mid] < target THEN
            left = mid + 1
        ELSE
            right = mid - 1
        END IF
    END WHILE

    return FALSE, CURRENT_TIME() - start_time
END FUNCTION
```

Grafik



D. Analisis perbandingan kedua algoritma

1. Pencarian Sekuensial:
   - Efisien untuk data yang tidak terurut.
   - Waktu eksekusi meningkat secara linier dengan bertambahnya jumlah data.
   - Kurang efisien untuk jumlah data yang sangat besar karena kompleksitas waktu O(n).
2. Pencarian Biner:
   - Memerlukan data yang sudah terurut.
   - Lebih efisien untuk jumlah data yang sangat besar karena kompleksitas waktu O(log n).
   - Waktu eksekusi meningkat secara logaritmik dengan bertambahnya jumlah data.

E. Referensi

1. YouTube. (2023). *Presentasi Tugas Besar Kelompok 6 | Analisis Kompleksitas Algoritma - Telkom University* oleh Helmi Efendi Lubis, Raisya Azzuhra, Naufal Hanif Arsyah. https://www.youtube.com/watch?v=yWcQ45I9-V8
2. Otniel113. (2024). *Membandingkan Kompleksitas Algoritma Sorting: BubbleSort dan MergeSort*. https://github.com/Otniel113/TugasBesarAKA

F. Kesimpulan

Dari analisis dan hasil eksperimen, dapat disimpulkan bahwa pencarian biner lebih efisien dibandingkan pencarian sekuensial dalam konteks data yang terurut dan ukuran input yang besar. Sebaliknya, pencarian sekuensial lebih cocok untuk data yang tidak terurut atau ukuran input yang kecil hingga menengah.