TUGAS BESAR ANALISIS KOMPLEKSITAS ALGORITMA

Penghitungan Penghasilan Bulanan Karyawan

Mata Kuliah: Analisis Kompleksitas Algoritma

Dosen Pengampu : Rimba W Ciptasari

Disusun Oleh:

Rosalina Siagian 103012300324

Annisa Azzahratul Ilmi 103012300434

PROGRAM STUDI S1 INFORMATIKA

FAKULTASINFORMATIKA

2024

1. Pendahuluan

Deskripsi Studi Kasus

Studi kasus ini bertujuan untuk menganalisis efisiensi algoritma dalam menghitung total penghasilan bulanan karyawan yang terdiri dari berbagai komponen, seperti gaji pokok, tunjangan, dan bonus. Penghitungan total penghasilan ini dilakukan menggunakan dua metode algoritmatik: **Interatif** dan **Rekursif.**

- **Metode Iteratif** menghitung total penghasilan dengan menggunakan perulangan melalui seluruh data karyawan.
- **Metode Rekursif** memecah proses penghitungan menjadi bagian-bagian kecil secara bertahap hingga mendapatkan hasil akhir.

Simulasi dilakukan dengan menggunakan dataset karyawan dalam jumlah tertentu, mulai dari puluhan hingga ribuan data, untuk menganalisis performa algoritma pada skenario berikut:

- **Best Case (Kasus Terbaik):** Data yang sangat terstruktur atau sederhana.
- **Worst Case (Kasus Terburuk):** Data dengan distribusi komponen penghasilan yang bervariasi.
- **Average Case (Kasus Rata-Rata):** Data dalam jumlah besar yang memerlukan waktu komputasi tinggi.

Dataset karyawan dihasilkan secara acak untuk mencakup atribut berikut:

- **Karyawan_ID**: Identifikasi unik untuk setiap karyawan.
- **Gaji Pokok**: Komponen utama penghasilan.
- **Tunjangan**: Komponen tambahan penghasilan.
- **Bonus**: Penghasilan tambahan berdasarkan performa.
- **Potongan**: Pengurangan dari total penghasilan.

Output dari simulasi meliputi:

- Total penghasilan bulanan untuk setiap karyawan.
- Perbandingan waktu eksekusi antara metode Iteratif dan Rekursif.

Tujuan dari studi ini adalah untuk memahami kelebihan dan kekurangan metode Iteratif dan Rekursif dalam menghitung penghasilan karyawan, serta menentukan metode mana yang paling efisien dan tepat digunakan berdasarkan jumlah data dan kebutuhan sistem penggajian perusahaan.

Sebagai contoh:

Jika terdapat dataset karyawan sebagai berikut:

- Jumlah Karyawan: 4
- Dataset karyawan:
  [
  {ID: KRY0001, Gaji Pokok: 5.000.000, Tunjangan: 2.000.000, Bonus: 1.000.000, Potongan: 500.000},
  {ID: KRY0002, Gaji Pokok: 5.500.000, Tunjangan: 1.500.000, Bonus: 1.000.000, Potongan: 500.000},
  {ID: KRY0003, Gaji Pokok: 7.000.000, Tunjangan: 2.500.000, Bonus: 2.000.000, Potongan: 1.000.000},
  {ID: KRY0004, Gaji Pokok: 6.000.000, Tunjangan: 2.500.000, Bonus: 1.500.000, Potongan: 800.000}
  ]
- Hasil Simulasi:
  - Total Penghasilan (Iteratif): 3.349.348.934
  - Waktu Eksekusi (Iteratif): 0,017445 detik
  - Total Penghasilan (Rekursif): 3.349.348.934
  - Waktu Eksekusi (Rekursif): 0,018347 detik
- Tabel Perbandingan Waktu Eksekusi:
  - Metode Iteratif memiliki waktu eksekusi yang sedikit lebih cepat dibandingkan dengan Rekursif

2. Pembahasan

a. Deskripsi Dua Algoritma yang Dipilih untuk Menyelesaikan Permasalahan

Berikut adalah penjelasan dari algoritma iteratif dan rekursif yang digunakan dalam menyelesaikan studi kasus penghitungan penghasilan bulanan karyawan.

i. Algotima Iteratif

Algoritma iteratif menggunakan perulangan (loop) untuk menghitung total penghasilan bulanan setiap karyawan. Penghitungan dilakukan dengan menjumlahkan gaji pokok, tunjangan, bonus, dan mengurangi potongan untuk setiap karyawan dalam data.

Langkah-langkah:

1. Inisialisasi: Mulai dari elemen pertama data karyawan.
2. Proses Penghitungan:
   a. Ambil data gaji pokok, tunjangan, bonus, dan potongan untuk karyawan.
   b. Hitung total penghasilan: (Gaji Pokok + Tunjangan + Bonus - Potongan).
   c. Tambahkan hasil penghitungan ke total keseluruhan penghasilan.

3. Penanganan Data Kosong: Jika tidak ada data, algoritma akan langsung memberikan hasil 0.
4. Tampilkan hasil: Total penghasilan semua karyawan dihitung dan ditampilkan.

```python
# Fungsi iteratif untuk menghitung total penghasilan
@st.cache
def hitung_iteratif(data):
    total_penghasilan = 0
    for index, row in data.iterrows():
        total_penghasilan += row['Gaji_Pokok'] + row['Tunjangan'] + row['Bonus'] - row['Potongan']
    return total_penghasilan
```

ii. Algotirma Rekursif

Algoritma rekursif memecahkan masalah perhitungan menjadi bagian–bagian kecil yang diselesaikan secara bertahap hingga mencapai akhir data. Pada setiap tahap, fungsi memanggil dirinya sendiri dengan indeks data yang lebih kecil. Langkah-langkah:

1. Inisialisasi: Mulai dari indeks pertama data karyawan.
2. Proses Rekursif:
   a. Jika indeks terakhir tercapai, kembalikan nilai 0 sebagai dasar rekursi.
   b. Jika belum, ambil data gaji pokok, tunjangan, bonus, dan potongan untuk karyawan pada indeks tersebut.
   c. Hitung penghasilan karyawan tersebut, lalu tambahkan dengan hasil rekursif berikutnya.
3. Penanganan Data Kosong: Jika data kosong, algoritma langsung kembali dengan hasil 0 tanpa proses lebih lanjut.
4. Tampilkan Hasil: Total penghasilan seluruh karyawan dihitung dan ditampilkan setelah semua rekursif selesai.

```python
# Fungsi rekursif untuk menghitung total penghasilan
def hitung_rekursif(data, index=0):
    if index == len(data):
        return 0
    row = data.iloc[index]
    return (row['Gaji_Pokok'] + row['Tunjangan'] + row['Bonus'] - row['Potongan']) + hitung_rekursif(data, index + 1)
```

b. Analisis Efisiensi Algoritma

1. Iteratif

Algoritma iteratif menghitung total penghasilan karyawan dengan menjumlahkan komponen-komponen gaji (gaji pokok, tunjangan, bonus) menggunakan perulangan (loop). Setiap baris data karyawan diproses secara berurutan, dan total penghasilan dihitung dengan menambahkan nilai setiap komponen dan mengurangkan potongan.

- **Best Case (Kasus Terbaik)**

- Kasus terbaik terjadi jika setiap karyawan hanya memiliki satu komponen gaji yang perlu dihitung, misalnya hanya gaji pokok tanpa tunjangan, bonus, atau potongan.
- Dalam skenario ini, perulangan hanya berjalan sekali untuk menghitung total penghasilan.
- **Kompleksitas Waktu (C(n))**: O(1), karena waktu eksekusi tidak bergantung pada jumlah data karyawan.

- **Worst Case (Kasus Teburuk)**
  - Kasus terburuk terjadi ketika setiap karyawan memiliki beberapa komponen gaji (gaji pokok, tunjangan, bonus) dan potongan, sehingga setiap elemen data karyawan harus diproses sepenuhnya.
  - Dalam situasi ini, loop berjalan sebanyak jumlah karyawan (n) untuk menghitung total penghasilan semua karyawan.
  - **Kompleksitas Waktu (C(n))**: O(n), karena waktu eksekusi meningkat seiring bertambahnya jumlah data.
- **Average Case (Kasus Rata-Rata)**
  - Pada kasus rata-rata, setiap karyawan memiliki beberapa komponen gaji, namun jumlahnya tidak sebanyak pada kasus terburuk. Rata-rata, loop berjalan sebanyak setengah dari jumlah data karyawan.
  - **Kompleksitas Waktu (C(n))**: O(n), karena tetap memproses semua data, meskipun sebagian besar operasinya lebih ringan dibandingkan kasus terburuk.

Total Time Complexity:

- **Best Case**:
  $T(n) = k \cdot 1 \rightarrow O(1)$
  Pada base case, algoritma hanya membutuhkan satu langkah untuk menyelesaikan perhitungan. Ini terjadi jika hanya ada satu elemen data yang perlu dihitung, misalnya hanya gaji pokok.
- **Worst Case**:
  $T(n) = k \cdot n \rightarrow O(n)$
  Pada worst case, algoritma harus memproses semua data karyawan tanpa terkecuali. Waktu yang dibutuhkan akan bertambah seiring bertambahnya jumlah data karyawan
- **Average Case**:
  $T(n) = k \cdot \frac{n}{2} \rightarrow O(n)$
  Pada average case, algoritma memproses sekitar setengah dari total data karyawan. Meski begitu, waktu eksekusi tetap linear karena setiap data pada akhirnya diperiksa.

Penjelasan Kompleksitas

- **Best Case (O(1)):**
  Pada kasus ini, hanya sedikit data yang diproses, sehingga waktu eksekusi tetap cepat dan tidak tergantung pada jumlah data.
- **Worst Case (O(n)):**
  Semua data dalam dataset harus diperiksa satu per satu, sehingga waktu eksekusi meningkat seiring bertambahnya jumlah data.
- **Average Case (O(n)):**
  Dalam kasus rata-rata, algoritma memproses sebagian besar data, tetapi waktu eksekusi tetap linear karena banyaknya elemen yang terlibat.

2. Rekursif

Algoritma rekursif menghitung total penghasilan dengan membagi masalah besar menjadi masalah-masalah kecil yang diselesaikan secara bertahap. Setiap kali fungsi dipanggil, algoritma memproses satu elemen data karyawan (gaji pokok, tunjangan, bonus, dan potongan), lalu memanggil dirinya sendiri untuk elemen berikutnya hingga seluruh data selesai dihitung.

- **Best Case (Kasus Terbaik):**
  - Pada best case, hanya ada satu elemen data yang perlu dihitung, misalnya hanya gaji pokok tanpa tunjangan, bonus, atau potongan.
  - Fungsi rekursif hanya dipanggil satu kali untuk memproses elemen tersebut.
  - **Kompleksitas Waktu (C(n))**: O(1), karena waktu eksekusi tetap dan tidak tergantung pada jumlah data.
- **Worst Case (Kasus Terburuk):**
  - Pada worst case, semua elemen data karyawan (gaji pokok, tunjangan, bonus, dan potongan) harus dihitung satu per satu.
  - Fungsi rekursif akan dipanggil sebanyak jumlah elemen data, yaitu n kali.
  - Setiap pemanggilan fungsi membutuhkan waktu yang sama (konstan), tetapi jumlah total pemanggilan meningkat sesuai dengan jumlah data.
  - **Kompleksitas Waktu (C(n))**: O(n), karena waktu eksekusi bertambah linear dengan jumlah data.
- **Average Case (Kasus Rata-Rata):**
  - Pada average case, fungsi rekursif dipanggil untuk sekitar setengah dari total elemen data sebelum mencapai kondisi akhir (base case).
  - Walaupun jumlah pemanggilan lebih sedikit dibandingkan kasus terburuk, kompleksitas waktu tetap linear karena mayoritas data diproses.
  - **Kompleksitas Waktu (C(n))**: O(n), karena setiap elemen secara tidak langsung terlibat dalam proses perhitungan.

Penjelasan Kompleksitas Waktu:

Best Case:

$T(n) = k \cdot 1 \rightarrow O(1)$

Pada kasus terbaik, hanya ada satu elemen data yang perlu dihitung, sehingga fungsi rekursif hanya dipanggil satu kali. Waktu eksekusi tetap konstan dan tidak bergantung pada jumlah data.

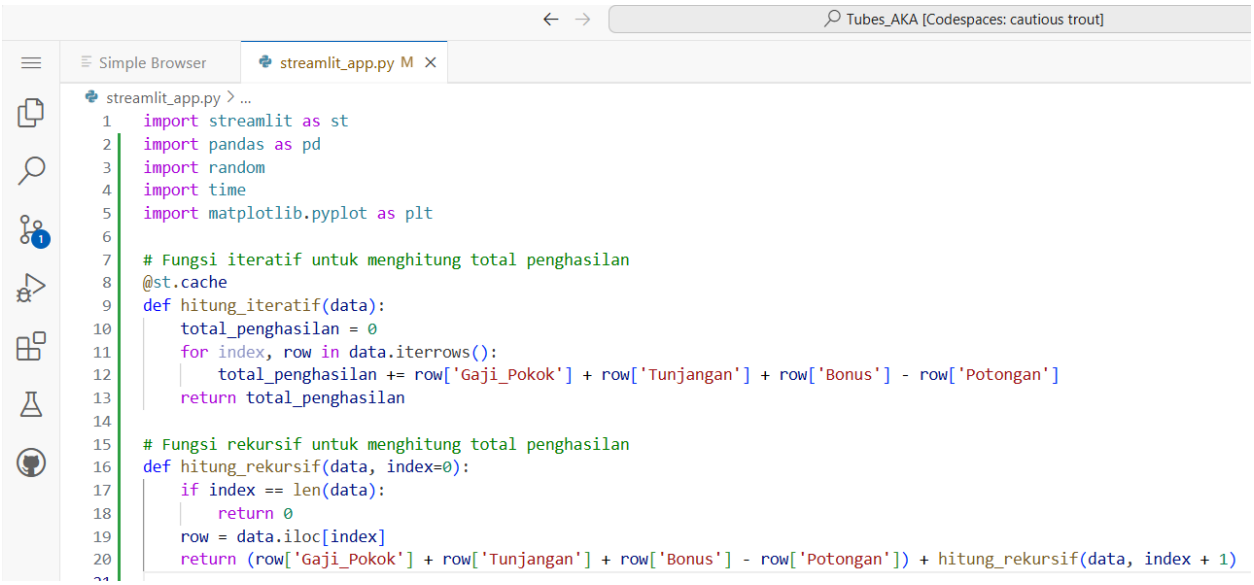
- **Worst Case:**
  $T(n) = k \cdot n \rightarrow O(n)$
  Pada kasus terburuk, setiap elemen data karyawan (gaji pokok, tunjangan, bonus, dan potongan) harus dihitung, sehingga fungsi dipanggil sebanyak jumlah elemen data (*nn*n). Waktu eksekusi meningkat linear seiring bertambahnya jumlah data.
- **Average Case:**
  $T(n) \approx k \cdot \frac{n}{2} \quad \rightarrow \quad O(n)$
  Pada rata-rata kasus, fungsi rekursif dipanggil untuk sekitar setengah dari total elemen data sebelum mencapai kondisi akhir (base case). Namun, kompleksitas waktu tetap linear karena sebagian besar data diproses.

Tabel Perbandingan Iteratif dan Rekursif

Kriteria	Iteratif	Rekursif
Implementasi	Lebih mudah ditulis dengan loop biasa. Kode umumnya lebih sederhana dan langsung	Membutuhkan pemahaman tentang rekursi. Kode bisa lebih ringkas tapi perlu kondisi dasar yang tepat
Kompleksitas Waktu	Umumnya O(n), efisien untuk data kecil-menengah	Juga O(n), tapi ada overhead dari pemanggilan fungsi berulang
Efisiensi Memori	Lebih hemat memori karena tidak menyimpan panggilan fungsi dalam stack	Memakan lebih banyak memori karena setiap panggilan rekursif ditumpuk dalam memory stack
Debugging	Lebih mudah dilacak karena alur program yang linear dan bisa diperiksa step by step	Lebih sulit di-debug karena banyak panggilan fungsi bertumpuk dan sulit melacak state program
Kecenderungan Error	Risiko error lebih kecil, biasanya hanya infinite loop yang perlu diwaspadai	Rawan stack overflow jika rekursi terlalu dalam atau kondisi dasar tidak tepat

3. Grafik dan Tabel Perbandingan Running Time

Kode Program


Tubes_AKA [Codespaces: cautious trout]
Simple Browser
streamlit_app.py M
streamlit_app.py > ...
1 import streamlit as st
2 import pandas as pd
3 import random
4 import time
5 import matplotlib.pyplot as plt
6
7 # Fungsi iteratif untuk menghitung total penghasilan
8 @st.cache
9 def hitung_iteratif(data):
10 total_penghasilan = 0
11 for index, row in data.iterrows():
12 total_penghasilan += row['Gaji_Pokok'] + row['Tunjangan'] + row['Bonus'] - row['Potongan']
13 return total_penghasilan
14
15 # Fungsi rekursif untuk menghitung total penghasilan
16 def hitung_rekursif(data, index=0):
17 if index == len(data):
18 return 0
19 row = data.iloc[index]
20 return (row['Gaji_Pokok'] + row['Tunjangan'] + row['Bonus'] - row['Potongan']) + hitung_rekursif(data, index + 1)

```python
# Simulasi data karyawan
def generate_data(jumlah):
    data = {
        "Karyawan_ID": [f"KRY{str(i).zfill(4)}" for i in range(1, jumlah + 1)],
        "Gaji_Pokok": [random.randint(2000000, 10000000) for _ in range(jumlah)],
        "Tunjangan": [random.randint(500000, 3000000) for _ in range(jumlah)],
        "Bonus": [random.randint(0, 2000000) for _ in range(jumlah)],
        "Potongan": [random.randint(0, 1000000) for _ in range(jumlah)],
    }
    return pd.DataFrame(data)

# Streamlit antarmuka utama
st.title("Penghitungan Penghasilan Bulanan Karyawan")

# Input jumlah karyawan
jumlah_karyawan = st.number_input("Masukkan jumlah karyawan:", min_value=10, max_value=10000, value=100)

# Inisialisasi variabel
if 'data_karyawan' not in st.session_state:
    st.session_state.data_karyawan = generate_data(jumlah_karyawan)
if 'time_iterative' not in st.session_state:
    st.session_state.time_iterative = 0
if 'time_recursive' not in st.session_state:
    st.session_state.time_recursive = 0

# Tombol untuk memulai perhitungan
if st.button("Hitung Total Penghasilan"):
    # Generate data karyawan
    st.session_state.data_karyawan = generate_data(jumlah_karyawan)
```

```
    # Perhitungan iteratif
    start_time_iterative = time.time()
    total_iterative = hitung_iteratif(st.session_state.data_karyawan)
    st.session_state.time_iterative = time.time() - start_time_iterative

    # Perhitungan rekursif
    start_time_recursive = time.time()
    total_recursive = hitung_rekursif(st.session_state.data_karyawan)
    st.session_state.time_recursive = time.time() - start_time_recursive

    # Tampilkan hasil
    st.write("### Hasil Perhitungan")
    st.write(f"Total Penghasilan (Iteratif): {total_iterative:,}")
    st.write(f"Waktu Eksekusi (Iteratif): {st.session_state.time_iterative:.6f} detik")
    st.write(f"Total Penghasilan (Rekursif): {total_recursive:,}")
    st.write(f"Waktu Eksekusi (Rekursif): {st.session_state.time_recursive:.6f} detik")

# Tampilkan tabel data
st.write("### Data Karyawan")
st.dataframe(st.session_state.data_karyawan)

# Membuat tabel perbandingan waktu eksekusi
comparison_data = {
    "Metode": ["Iteratif", "Rekursif"],
    "Waktu Eksekusi (detik)": [st.session_state.time_iterative, st.session_state.time_recursive]
}
comparison_df = pd.DataFrame(comparison_data)
st.write("### Tabel Perbandingan Waktu Eksekusi")
st.dataframe(comparison_df)
```

```python
# Membuat grafik perbandingan waktu eksekusi
if st.session_state.time_iterative > 0 and st.session_state.time_recursive > 0:
    n_values = range(100, jumlah_karyawan + 1, 100)
    iterative_times = []
    recursive_times = []

    for n in n_values:
        data_sample = generate_data(n)

        start_time_iter = time.time()
        hitung_iteratif(data_sample)
        iterative_times.append(time.time() - start_time_iter)

        start_time_rec = time.time()
        hitung_rekursif(data_sample)
        recursive_times.append(time.time() - start_time_rec)

    # Plot grafik
    fig, ax = plt.subplots()
    ax.plot(n_values, iterative_times, marker='o', label='Iteratif', color='blue')
    ax.plot(n_values, recursive_times, marker='o', label='Rekursif', color='red')
    ax.set_xlabel("Nilai n")
    ax.set_ylabel("Waktu Eksekusi (detik)")
    ax.set_title("Perbandingan Waktu Eksekusi Iteratif vs Rekursif")
    ax.legend()
    st.pyplot(fig)
```

Hasil Perhitungan

Penghitungan Penghasilan Bulanan Karyawan

Masukkan jumlah karyawan:

400 − +

Hitung Total Penghasilan

Hasil Perhitungan

Total Penghasilan (Iteratif): 3,294,770,957

Waktu Eksekusi (Iteratif): 0.017949 detik

Total Penghasilan (Rekursif): 3,294,770,957

Waktu Eksekusi (Rekursif): 0.019062 detik

Tabel Perbandingan Waktu Ekseskusi

Tabel Perbandingan Waktu Eksekusi

	Metode	Waktu Eksekusi (detik)
0	Iteratif	0.0179
1	Rekursif	0.0191

Grafik

4. Kesimpulan

Dalam studi kasus ini, dua metode algoritma, yaitu iteratif dan rekursif, digunakan untuk menghitung total penghasilan bulanan karyawan dengan mempertimbangkan komponen gaji seperti gaji pokok, tunjangan, bonus, dan potongan. Hasil analisis menunjukkan:

1. Efisiensi Waktu:
   a. Metode iteratif memiliki waktu eksekusi yang lebih cepat dibandingkan metode rekursif pada dataset besar.
   b. Kompleksitas waktu untuk kedua metode adalah **O(1)** pada kasus terbaik dan **O(n)** pada kasus terburuk maupun rata-rata.
2. Kompleksitas Algoritma:
   a. Metode iteratif lebih sederhana dalam implementasi dan cocok untuk data yang besar karena tidak memiliki overhead pemanggilan fungsi.
   b. Metode rekursif membagi permasalahan menjadi sub-masalah lebih kecil tetapi cenderung memerlukan lebih banyak memori karena pemanggilan fungsi bertumpuk.

3. Simulasi dan Hasil:
   a. Total penghasilan bulanan yang dihitung menggunakan kedua metode menghasilkan nilai yang sama.

b. Waktu eksekusi iteratif sedikit lebih cepat dibandingkan rekursif pada dataset besar, menunjukkan bahwa iteratif lebih efisien dalam konteks ini.

4. Rekomendasi:
   a. Untuk penghitungan penghasilan karyawan dalam sistem penggajian perusahaan dengan dataset besar, metode iteratif lebih disarankan karena efisiensi waktu dan penggunaan memori yang lebih rendah.
   b. Metode rekursif dapat digunakan untuk dataset kecil atau situasi yang memerlukan pendekatan solusi berbasis pemecahan sub-masalah.

Studi ini memberikan wawasan tentang pemilihan algoritma berdasarkan kebutuhan sistem dan ukuran dataset, mendukung pengambilan keputusan yang optimal dalam sistem penggajian perusahaan.

5. Referensi

https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/

https://www.geeksforgeeks.org/analysis-of-different-sorting-techniques/?ref=header_outind

https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/?ref=header_outind

https://webzid.com/rekursif-vs-iteratif/