Tugas Besar Strategi Algoritma

Perbandingan Algoritma Sorting Dalam Pengurutan Revenue pada dataset Restoran

Daffa Hauzananda Arsyah (2211102169)

Mario Firdaus Abdillah (2211102296)

Moh. Hikam Abdul Karim (2211102278)

Alif Irsyad Santoso (2211102315)

Arief Budi Mulyawan (2211102287)

PROGRAM STUDI TEKNIK INFORMATIKA

FAKULTAS INFORMATIKA

UNIVERSITAS TELKOM

PURWOKERTO

2024

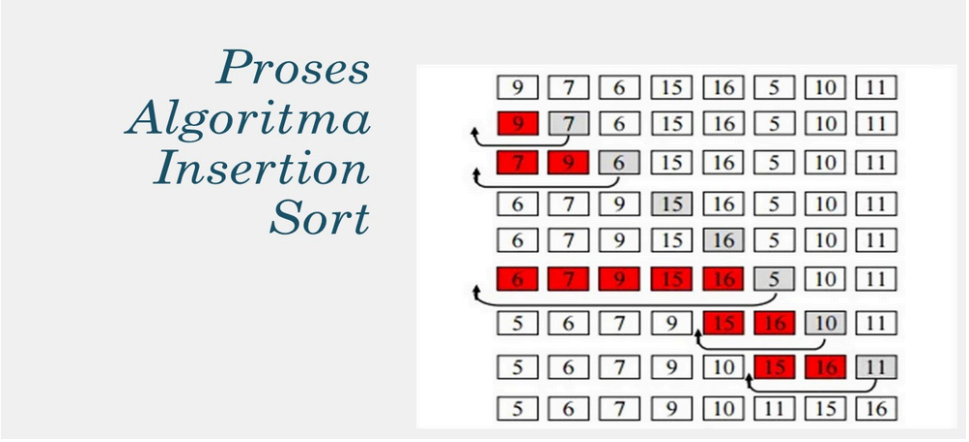
Dasar Teori

Sorting

algoritma pengurutan adalah algoritma yang menempatkan elemen-elemen daftar ke dalam urutan. Urutan yang paling sering digunakan adalah urutan numerik dan urutan leksikografis, baik menaik atau menurun, Penyortiran yang efisien penting untuk mengoptimalkan efisiensi algoritma lain (seperti algoritma pencarian dan penggabungan) yang memerlukan data masukan dalam daftar yang diurutkan.

Insertion Sort

Algoritma insertion sort, adalah metode pengurutan dengan cara menyisipkan elemen data pada posisi yang tepat. Pencarian posisi yang tepat dilakukan dengan melakukan pencarian berurutan didalam barisan elemen, selama pencarian posisi yang tepat dilakukan pergeseran elemen. Pengurutan insertion sort sangat mirip dengan konsep permainan kartu, bahwa setiap kartu disisipkan secara berurutan dari kiri ke kanan sesuai dengan besar nilai kartu tersebut, dengan syarat apabila sebuah kartu disisipkan pada posisi tertentu kartu yang lain akan bergeser maju atau mundur sesuai dengan besaran nilai yang dimiliki.



(Mengenal Sorting Berserta Contoh Source Code Pada Struktur Data - Daisma Bali)

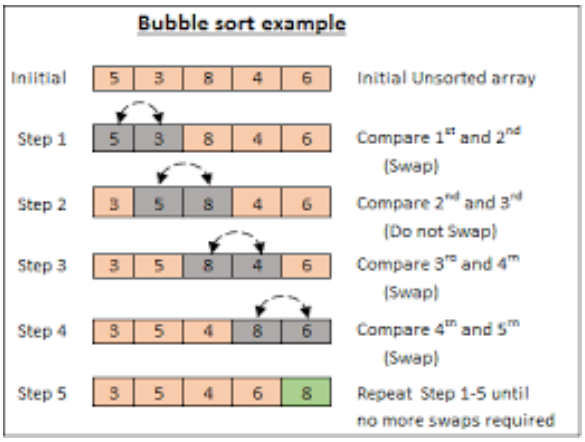
```
INSERT(A, i)
1. j ← i − 1
2. v ← A[i]
3. while (j ≥ 0 and A[j] > v) do
4.         A[j + 1] ← A[j]
5.         j ← j − 1
6. A[j + 1] ← v

INSERTIONSORT
1. for i ← 1 to n − 1 do
2.         INSERT(A, i)
```

(Pseudocode Insertion Sort oleh Faro S. Marino (2020)

Bubble Sort

Bubble sort adalah salah satu jenis algoritma sorting. Ide dari algoritma ini adalah untuk mengulangi proses perbandingan antara setiap elemen array dan menukarnya jika tidak berurutan. Penyelarasan elemen-elemen ini diulangi sampai tidak diperlukan penggantian lebih lanjut. Algoritma ini termasuk dalam kelompok algoritma pengurutan komparatif karena menggunakan perbandingan dalam operasi antar elemen.



(Mekanisme bubble sort)

```
void BubbleSort(Vector a, int n)
{
    for(int j=n-1; j > 0; j--)
        for(int k=0; k < j; k++)
            if (a[k+1] < a[k])
                Swap(a,k,k+1);
}
```

Quick Sort

Metode Quick sort merupakan suatu metode yang paling cepat dalam proses pengurutan data. Quick sort sering disebut juga metode partisi (partition exchange sort). Metode ini diperkenalkan pertama kali oleh C.A.R. Hoare pada tahun 1962. Untuk mempertinggi efektifitas dari metode ini, digunakan teknik menukarkan dua elemen dengan jarak yang cukup besar. Quick sort merupakan sebuah algoritma sorting dari model divide dan conquer. Divide dan conquer adalah metode pemecahan masalah yang bekerja dengan membagi masalah menjadi beberapa sub-masalah yang lebih kecil, kemudian menyelesaikan masing-masing sub-masalah secara independen, dan akhirnya menggabungkannya. Implementasi dalam tugas besar kali ini akan mengadaptasi model pseudocode dari Joseph Lala (1999), *A perspective on quickshort*.

```
procedure partition(A, l, u)

begin

Step 1. Set i = l; j = r + 1; pivot = A[l];

Step 2. while(true){
          while(A[++i] < pivot);
          while(A[--j] > pivot);
          if i < j then exchange A[i] and A[j];
             else break;
             }
          A[l] = A[j];
          A[j] = pivot;

end procedure
```

(pseudocode quick sort oleh Joseph Lala)

Implementasi

Tahap implementasi mengandung proses penulisan source code dan eksekusinya. Keterangan mengenai program akan tertulis dalam komentar, yang mana mengandung fungsi bagan program, library yang program gunakan, serta keterangan data pengujian. Berikut merupakan spesifikasi dari perangkat pengujian

- RAM : 12.7 GB
- CPU : Intel Xeon CPU./ 2 vCPU
- Operating System : Linux Ubuntu
- Interpreter : CPython
- IDE : Jupyter Notebook dalam Google Collab

Insertion Sort

Pseudocode:

```
BEGIN
    IMPORT libraries: Pandas, time, Matplotlib

    // Memuat dataset
    DEFINE file_path AS 'restaurant_data.csv'
    LOAD restaurant_data FROM file_path


    // Mendefinisikan fungsi Insertion Sort
    FUNCTION insertion_sort(arr):
        FOR i FROM 1 TO LENGTH(arr) - 1 DO
            key ← arr[i]
            j ← i - 1
            WHILE j >= 0 AND arr[j] > key DO
                arr[j + 1] ← arr[j]
                j ← j - 1
            END WHILE
            arr[j + 1] ← key
        END FOR
        RETURN arr
    END FUNCTION
```

```
    // Menentukan ukuran data untuk diuji dan mempersiapkan
pengujian
    DEFINE sizes AS [10, 200, 1000, 3000, 7000]
    DEFINE runtimes AS empty list


    // Memeriksa apakah kolom 'Revenue' ada
    IF 'Revenue' EXISTS IN restaurant_data THEN
        FOR size IN sizes DO
            IF size <= LENGTH(restaurant_data) THEN
                // Mengambil data uji dan mengukur waktu eksekusi
                test_data ← FIRST size ELEMENTS OF
restaurant_data['Revenue']
                start_time ← CURRENT TIME
                CALL insertion_sort(test_data)
                elapsed_time ← CURRENT TIME - start_time
                ADD elapsed_time TO runtimes
                PRINT "Ukuran Input:", size, "Waktu Eksekusi:",
elapsed_time
            ELSE
                PRINT "Ukuran melebihi ukuran dataset:", size
            END IF
        END FOR
    ELSE
        PRINT "Kolom 'Revenue' tidak ada dalam dataset."
    END IF


    // Membuat grafik hasil
    INITIALIZE a plot figure
    PLOT sizes AGAINST runtimes WITH LABEL 'Insertion Sort
Runtime'
    SET judul grafik, label sumbu, dan grid
    LIMIT rentang sumbu y HINGGA 10 detik
    DISPLAY the plot
END

BEGIN
    IMPORT libraries: Pandas, time, Matplotlib

    // Memuat dataset
    DEFINE file_path AS 'restaurant_data.csv'
```

```
    LOAD restaurant_data FROM file_path


    // Mendefinisikan fungsi Insertion Sort
    FUNCTION insertion_sort(arr):
        FOR i FROM 1 TO LENGTH(arr) - 1 DO
            key ← arr[i]
            j ← i - 1
            WHILE j >= 0 AND arr[j] > key DO
                arr[j + 1] ← arr[j]
                j ← j - 1
            END WHILE
            arr[j + 1] ← key
        END FOR
        RETURN arr
    END FUNCTION


    // Menentukan ukuran data untuk diuji dan mempersiapkan
pengujian
    DEFINE sizes AS [10, 200, 1000, 3000, 7000]
    DEFINE runtimes AS empty list


    // Memeriksa apakah kolom 'Revenue' ada
    IF 'Revenue' EXISTS IN restaurant_data THEN
        FOR size IN sizes DO
            IF size <= LENGTH(restaurant_data) THEN
                // Mengambil data uji dan mengukur waktu eksekusi
                test_data ← FIRST size ELEMENTS OF
restaurant_data['Revenue']
                start_time ← CURRENT TIME
                CALL insertion_sort(test_data)
                elapsed_time ← CURRENT TIME - start_time
                ADD elapsed_time TO runtimes
                PRINT "Ukuran Input:", size, "Waktu Eksekusi:",
elapsed_time
            ELSE
                PRINT "Ukuran melebihi ukuran dataset:", size
            END IF
        END FOR
    ELSE
```

```
        PRINT "Kolom 'Revenue' tidak ada dalam dataset."
    END IF



    // Membuat grafik hasil
    INITIALIZE a plot figure
    PLOT sizes AGAINST runtimes WITH LABEL 'Insertion Sort
Runtime'
    SET judul grafik, label sumbu, dan grid
    LIMIT rentang sumbu y HINGGA 10 detik
    DISPLAY the plot
END
```

Source Code:

```
# Mengimpor library yang diperlukan
import pandas as pd  # Pandas (https://pandas.pydata.org/) -
Digunakan untuk manipulasi data dan membaca file CSV.
import time  # Time (https://docs.python.org/3/library/time.html)
- Digunakan untuk mengukur waktu eksekusi algoritma.
import matplotlib.pyplot as plt  # Matplotlib
(https://matplotlib.org/) - Digunakan untuk membuat grafik
visualisasi data.

# Memuat dataset dari file CSV
file_path = 'restaurant_data.csv'  # Path file CSV
restaurant_data = pd.read_csv(file_path)  # Membaca data dari file
CSV

# Fungsi Insertion Sort untuk mengurutkan data
def insertion_sort(arr):
    # Iterasi dimulai dari indeks ke-1 (karena elemen pertama
dianggap sudah terurut)
    for i in range(1, len(arr)):
        key = arr[i]  # Menyimpan elemen yang akan disisipkan
        j = i - 1  # Menentukan posisi untuk mencari tempat elemen
key
        # Memindahkan elemen-elemen yang lebih besar dari key satu
posisi ke kanan
        while j >= 0 and arr[j] > key:
            arr[j + 1] = arr[j]  # Pindahkan elemen lebih besar ke
kanan
            j -= 1  # Periksa elemen sebelumnya
```

```
        arr[j + 1] = key  # Menyisipkan elemen key di posisi yang
benar
    return arr  # Mengembalikan hasil urutan

# Menentukan berbagai ukuran data untuk diuji
sizes = [10, 200, 1000, 3000, 7000]  # Ukuran-ukuran input data
yang akan diuji
runtimes = []  # Daftar untuk menyimpan waktu eksekusi untuk
setiap ukuran data

# Memeriksa apakah kolom 'Revenue' ada dalam dataset
if 'Revenue' in restaurant_data.columns:
    # Menguji algoritma pada berbagai ukuran data
    for size in sizes:
        # Jika ukuran yang diuji lebih kecil atau sama dengan
jumlah data
        if size <= len(restaurant_data):
            # Memilih subset data 'Revenue' untuk ukuran data yang
diuji
            test_data = restaurant_data['Revenue'][:size].tolist()
            # Mengukur waktu mulai eksekusi
            start_time = time.time()
            # Menjalankan algoritma Insertion Sort
            insertion_sort(test_data)
            # Menghitung waktu eksekusi
            elapsed_time = time.time() - start_time
            # Menyimpan waktu eksekusi ke dalam list
            runtimes.append(elapsed_time)
            # Mencetak waktu eksekusi untuk ukuran input tertentu
            print(f"Ukuran Input: {size}, Waktu yang dibutuhkan:
{elapsed_time:.6f} detik")
        else:
            # Jika ukuran lebih besar dari jumlah data dalam
dataset
            print(f"Ukuran: {size} melebihi ukuran dataset.")
else:
    # Menampilkan pesan jika kolom 'Revenue' tidak ada dalam
dataset
    print("Kolom 'Revenue' tidak ada dalam dataset.")

# Membuat grafik untuk menampilkan waktu eksekusi berdasarkan
ukuran data
```

```
plt.figure(figsize=(10, 6))  # Mengatur ukuran grafik
plt.plot(sizes, runtimes, marker='o', linestyle='-', color='r',
label='Waktu Eksekusi Insertion Sort')
# Menambahkan judul dan label untuk sumbu X dan Y
plt.title('Waktu Eksekusi Insertion Sort vs. Ukuran Data',
fontsize=14)
plt.xlabel('Ukuran Data', fontsize=12)
plt.ylabel('Waktu Eksekusi (detik)', fontsize=12)
plt.grid(True)  # Menambahkan grid pada grafik
plt.ylim(0, 10)  # Mengatur rentang waktu eksekusi (maksimum 10
detik)
plt.legend()  # Menampilkan legenda pada grafik
plt.show()  # Menampilkan grafik
```

Quick Sort

Pseudo Code:

```
BEGIN
    // Mengimpor pustaka yang diperlukan
    IMPORT libraries: Pandas, time, Matplotlib

    // Memuat dataset
    DEFINE file_path AS 'restaurant_data.csv'
    LOAD restaurant_data FROM file_path

    // Mendefinisikan fungsi Quick Sort
    FUNCTION quick_sort(arr):
        IF LENGTH(arr) <= 1 THEN
            RETURN arr  // Basis rekursi: array dengan 0 atau 1
elemen
        ELSE
            pivot ← arr[0]  // Elemen pertama sebagai pivot
            left ← [x FOR x IN arr IF x < pivot]  // Elemen lebih
kecil dari pivot
            middle ← [x FOR x IN arr IF x == pivot]  // Elemen
sama dengan pivot
            right ← [x FOR x IN arr IF x > pivot]  // Elemen lebih
besar dari pivot
            RETURN quick_sort(left) + middle + quick_sort(right)
// Rekursi pada left dan right
    END FUNCTION
```

```
    // Menentukan ukuran data untuk pengujian dan menyiapkan hasil
runtime
    DEFINE sizes AS [10, 200, 1000, 3000, 7000]
    DEFINE runtimes AS empty list

    // Memeriksa apakah kolom 'Revenue' ada dalam dataset
    IF 'Revenue' EXISTS IN restaurant_data THEN
        FOR size IN sizes DO
            IF size <= LENGTH(restaurant_data) THEN
                // Ekstraksi subset data 'Revenue'
                test_data ← FIRST size ELEMENTS OF
restaurant_data['Revenue']
                start_time ← CURRENT TIME  // Mulai pengukuran
waktu
                CALL quick_sort(test_data)  // Jalankan Quick Sort
pada data uji
                elapsed_time ← CURRENT TIME - start_time  //
Hitung waktu eksekusi
                ADD elapsed_time TO runtimes  // Simpan waktu
eksekusi ke daftar runtimes
                PRINT "Input Size:", size, "Execution Time:",
elapsed_time
            ELSE
                PRINT "Size exceeds dataset size:", size  //
Ukuran melebihi dataset
            END IF
        END FOR
    ELSE
        PRINT "Column 'Revenue' does not exist in dataset."  //
Kolom tidak ditemukan
    END IF

    // Membuat dan menampilkan grafik runtime
    INITIALIZE a plot figure  // Inisialisasi grafik
    PLOT sizes AGAINST runtimes WITH LABEL 'Quick Sort Runtime'
// Plot ukuran data vs runtime
    SET graph title, axis labels, and grid  // Tambahkan judul,
label sumbu, dan grid
    LIMIT y-axis range TO 10 seconds  // Batasi sumbu y hingga 10
detik
    DISPLAY the plot  // Tampilkan grafik
END
```

Source Code:

```
# Sumber pustaka:
# pandas: Digunakan untuk manipulasi data berbasis tabel.
# - Creator: Wes McKinney
# - Fungsi utama: Pemrosesan dan analisis data tabular.
# - URL: https://pandas.pydata.org/

# time: Modul bawaan Python untuk pengukuran waktu.
# - Creator: Guido van Rossum dan tim Python.
# - Fungsi utama: Mengukur waktu eksekusi dan manipulasi waktu.
# - URL: https://docs.python.org/3/library/time.html

# matplotlib.pyplot: Untuk membuat grafik dan visualisasi data.
# - Creator: John D. Hunter
# - Fungsi utama: Plotting data dalam berbagai format grafik.
# - URL: https://matplotlib.org/

import pandas as pd  # Untuk manipulasi data
import time  # Untuk mengukur waktu eksekusi
import matplotlib.pyplot as plt  # Untuk visualisasi data

# Memuat dataset restaurant dari file CSV
file_path = 'restaurant_data.csv'
restaurant_data = pd.read_csv(file_path)

# Fungsi QuickSort
def quick_sort(arr):
    """
    Fungsi untuk mengurutkan elemen dalam array menggunakan
algoritma QuickSort.
    - Jika panjang array <= 1, array langsung dikembalikan (basis
rekursi).
    - Pivot dipilih sebagai elemen pertama dari array.
    - Membagi array menjadi tiga bagian:
      left (elemen lebih kecil dari pivot),
      middle (elemen sama dengan pivot),
      dan right (elemen lebih besar dari pivot).
    - Memanggil quick_sort secara rekursif pada left dan right.
    """
    if len(arr) <= 1:
        return arr
```

```
    else:
        pivot = arr[0]  # Pivot dipilih sebagai elemen pertama
        left = [x for x in arr if x < pivot]  # Elemen lebih kecil
dari pivot
        middle = [x for x in arr if x == pivot]  # Elemen sama
dengan pivot
        right = [x for x in arr if x > pivot]  # Elemen lebih
besar dari pivot
        return quick_sort(left) + middle + quick_sort(right)

# Test algoritma quick sort dengan ukuran data yang berbeda
sizes = [10, 200, 1000, 3000, 7000]  # Daftar ukuran data yang
akan diuji
runtimes = []  # Untuk menyimpan waktu eksekusi

if 'Revenue' in restaurant_data.columns:  # Memeriksa apakah kolom
'Revenue' ada dalam dataset
    for size in sizes:  # Iterasi untuk setiap ukuran data
        if size <= len(restaurant_data):  # Memastikan ukuran
tidak melebihi panjang dataset
            # Ambil subset data 'Revenue' sesuai ukuran dan
konversi ke list
            test_data = restaurant_data['Revenue'][:size].tolist()
            start_time = time.time()  # Mulai pengukuran waktu
            quick_sort(test_data)  # Jalankan fungsi quick_sort
pada data uji
            elapsed_time = time.time() - start_time  # Hitung
waktu yang dibutuhkan
            runtimes.append(elapsed_time)  # Tambahkan waktu ke
daftar runtimes
            print(f"Size: {size}, Time taken: {elapsed_time:.6f}
seconds")
        else:
            print(f"Size: {size} exceeds the dataset size.")  #
Peringatan jika ukuran melebihi dataset
else:
    print("The column 'Revenue' does not exist in the dataset.")
# Pesan jika kolom tidak ditemukan

# Menampilkan hasil dalam grafik
plt.figure(figsize=(10, 6))  # Mengatur ukuran grafik
```

```
plt.plot(sizes, runtimes, marker='o', linestyle='-', color='b',
label='Quick Sort Runtime')
plt.title('Quick Sort Runtime vs. Data Size', fontsize=14)  #
Judul grafik
plt.xlabel('Data Size', fontsize=12)  # Label sumbu x
plt.ylabel('Runtime (seconds)', fontsize=12)  # Label sumbu y
plt.ylim(0, 10)  # Mengatur batas atas sumbu y menjadi 10 detik
plt.grid(True)  # Menambahkan grid pada grafik
plt.legend()  # Menambahkan legenda
plt.show()  # Menampilkan grafik
```

Bubble Sort

Pseudo Code:

```
BEGIN
    IMPORT libraries: Pandas, time, Matplotlib

    // Memuat dataset
    DEFINE file_path AS 'restaurant_data.csv'
    LOAD data FROM file_path

    // Menyiapkan data kolom 'Revenue'
    EXTRACT 'Revenue' COLUMN FROM data
    REMOVE missing values (NaN)
    CONVERT values TO integer
    STORE cleaned data AS revenue_data

    // Mendefinisikan algoritma Bubble Sort
    FUNCTION bubble_sort(arr):
        SET n AS LENGTH(arr)
        FOR i FROM 0 TO n-1 DO
            FOR j FROM 0 TO n-i-2 DO
                IF arr[j] > arr[j+1] THEN
                    SWAP arr[j] AND arr[j+1]
                END IF
            END FOR
        END FOR
    END FUNCTION

    // Menentukan ukuran input untuk pengujian
    DEFINE input_sizes AS [10, 200, 1000, 3000, 7000]
    DEFINE running_times AS empty list
```

```
    // Mengukur waktu eksekusi untuk setiap ukuran input
    FOR size IN input_sizes DO
        DEFINE test_data AS FIRST size ELEMENTS OF revenue_data

        // Menghitung waktu eksekusi
        SET start_time AS CURRENT TIME
        CALL bubble_sort(test_data)
        SET end_time AS CURRENT TIME

        CALCULATE runtime_seconds AS end_time - start_time
        APPEND (size, runtime_seconds) TO running_times

        PRINT "Ukuran input", size, ":", runtime_seconds, "detik"
    END FOR

    // Membuat grafik hubungan ukuran input dan waktu eksekusi
    INITIALIZE a plot figure
    PLOT input_sizes AGAINST running_times USING markers and line
style
    SET title AS 'Bubble Sort: Waktu Eksekusi vs Ukuran Input'
    LABEL x-axis AS 'Ukuran Input (n)'
    LABEL y-axis AS 'Waktu Eksekusi (detik)'
    SET y-axis LIMIT AS [0, 10]
    ADD grid TO plot
    DISPLAY plot

    // Menampilkan hasil analisis dalam tabel
    PRINT "Hasil Analisis Waktu Eksekusi:"
    PRINT " Ukuran Input  Waktu Eksekusi (detik)"
    FOR size, runtime IN running_times DO
        PRINT size, runtime
    END FOR
END
```

Source Code:

```
# pandas: Digunakan untuk manipulasi data berbasis tabel.
# - Fungsi utama: Pemrosesan dan analisis data tabular.
# - URL: https://pandas.pydata.org/
```

```python
import pandas as pd

# time: Modul bawaan Python untuk pengukuran waktu.
# - Fungsi utama: Mengukur waktu eksekusi dan manipulasi waktu.
# - URL: https://docs.python.org/3/library/time.html
import time

# matplotlib.pyplot: Untuk membuat grafik dan visualisasi data.
# - Fungsi utama: Plotting data dalam berbagai format grafik.
# - URL: https://matplotlib.org/
import matplotlib.pyplot as plt

# Memuat dataset dari file CSV
file_path = 'restaurant_data.csv'
data = pd.read_csv(file_path)

# Mengambil kolom 'Revenue' dari data dan memastikan nilai-nilai tersebut dalam format integer
# Menghilangkan data yang kosong (NaN) dan mengkonversi tipe data menjadi integer
revenue_data = data['Revenue'].dropna().astype(int).tolist()

# Mendefinisikan algoritma Bubble Sort
# Bubble Sort adalah algoritma pengurutan yang membandingkan elemen-elemen secara berpasangan dan menukarnya jika urutannya salah
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        # Melakukan perbandingan antara elemen yang berdekatan
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:  # Jika elemen sebelumnya lebih besar, tukar posisi
                arr[j], arr[j+1] = arr[j+1], arr[j]

# Menghasilkan ukuran input data yang berbeda untuk pengujian kecepatan algoritma
input_sizes = [10, 200, 1000, 3000, 7000]
running_times = []  # Menyimpan waktu eksekusi untuk setiap ukuran input

# Mengukur waktu eksekusi untuk setiap ukuran input
```

```
for size in input_sizes:
    # Mengambil subset data berdasarkan ukuran input saat ini
    test_data = revenue_data[:size]

    # Menghitung waktu sebelum dan sesudah eksekusi Bubble Sort
    start_time = time.time()
    bubble_sort(test_data)
    end_time = time.time()

    # Menghitung selisih waktu eksekusi dalam detik
    running_time_seconds = end_time - start_time
    running_times.append((size, running_time_seconds))

    # Menampilkan waktu eksekusi untuk ukuran input tertentu
    print(f"Ukuran input {size}: {running_time_seconds:.4f}
detik")

# Menampilkan grafik yang menunjukkan hubungan antara waktu
eksekusi dan ukuran input
plt.figure(figsize=(10, 6))  # Menentukan ukuran grafik
plt.plot(input_sizes, [rt[1] for rt in running_times], marker='o',
linestyle='-', color='b')
plt.title('Bubble Sort: Waktu Eksekusi vs Ukuran Input')  # Judul
grafik
plt.xlabel('Ukuran Input (n)')  # Label sumbu X
plt.ylabel('Waktu Eksekusi (detik)')  # Label sumbu Y
plt.ylim(0, 10)  # Menentukan batasan sumbu Y agar grafik lebih
jelas
plt.grid(True)  # Menambahkan grid pada grafik
plt.show()  # Menampilkan grafik

# Menampilkan analisis waktu eksekusi dalam format tabel
print("\nHasil Analisis Waktu Eksekusi:")
print(" Ukuran Input  Waktu Eksekusi (detik)")
# Menampilkan hasil waktu eksekusi untuk setiap ukuran input dalam
format tabel
for size, runtime in running_times:
    print(f"        {size:4d}            {runtime:10.6f}")
```

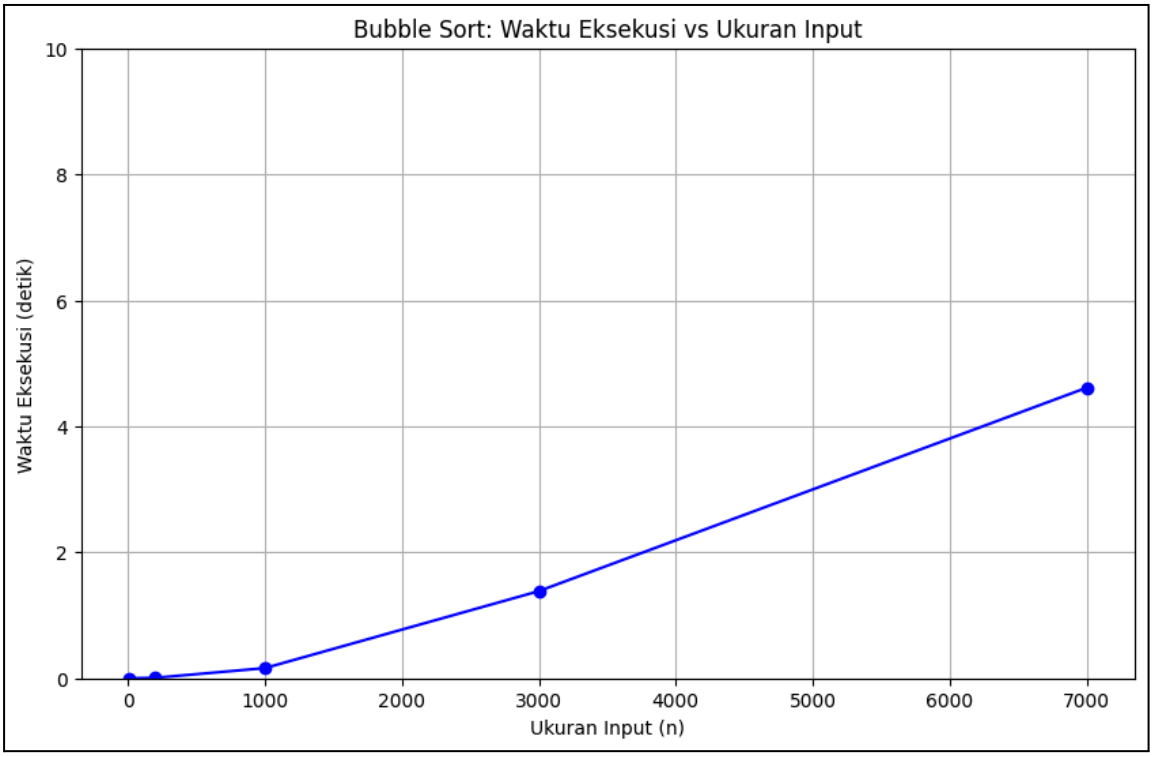
Pengujian

Data untuk pengujian berikut merupakan data set berisi revenue dari sebuah restoran berjudul "*Restaurant Revenue Prediction Dataset*". Data tersebut bersumber dari kaggle dan bersifat publik sehingga legal untuk tujuan akademik. Kami akan mengambil 7000 dari 8000 data sebagai sampel pengujian.

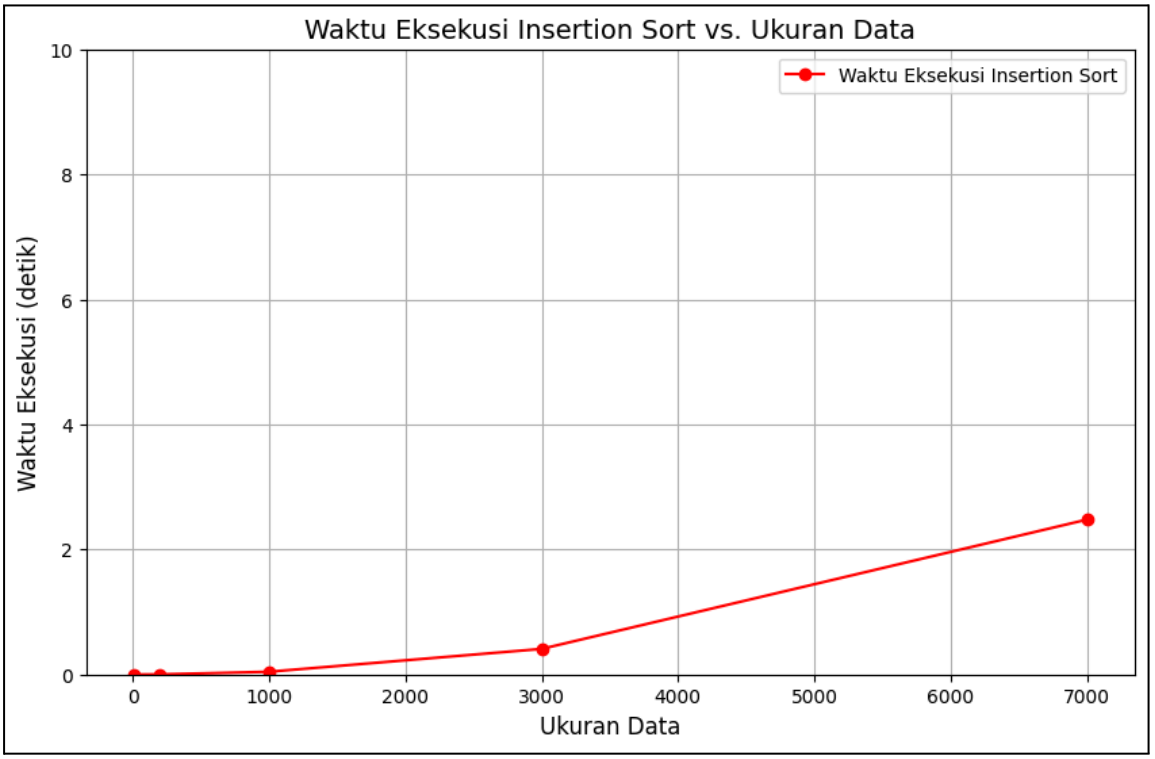
Untuk metode pengujian sendiri, program akan berjalan dan mengeksekusi ketiga algoritma sorting terhadap jumlah input data yang berbeda, mulai dari 10, 200, 1000 3000, hingga 7000, kemudian membandingkan run time antara masing - masing algoritma. Berikut merupakan hasil dari pengujian.

n	Waktu eksekusi Algoritma Bubble sort	Waktu eksekusi Algoritma Insertion sort	Waktu eksekusi Algoritma Quick sort
10	0.000023	0.000011	0.000019
200	0.008577	0.001533	0.000434
1000	0.162157	0.043676	0.002547
3000	1.383981	0.409279	0.009734
7000	4.612817	2.477953	0.021861

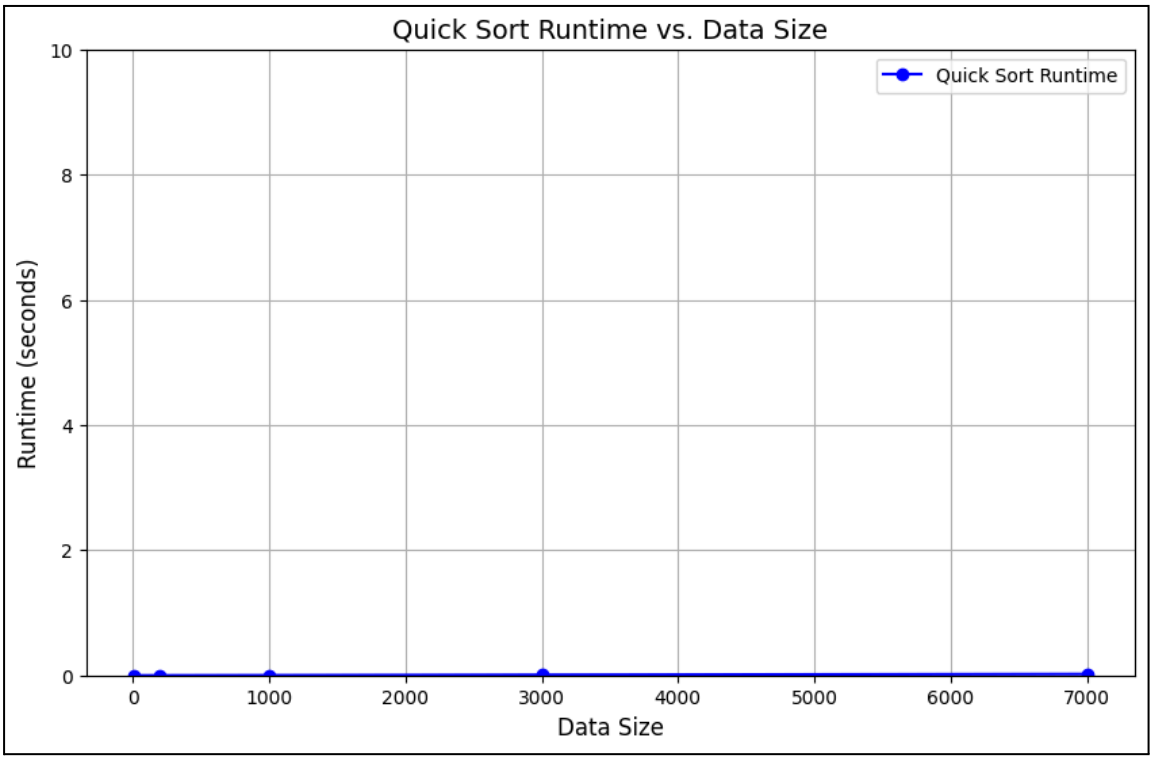
Jika kita bentuk menjadi graph untuk visualisasi data, adalah sebagai berikut:



(Grafik performa bubble sort)



(Grafik performa insertion sort)



(Grafik performa Quick sort)

Analisis Hasil Pengujian

Berdasarkan referensi dari berbagai jurnal; Bubble sort, dengan rata - rata kompleksitas sebagai $O(n^2)$ menurut Alake, R. (2024), Insertion sort, dengan rata - rata kompleksitas $O(n^2)$ oleh Faro (2020), dan Quick sort, dengan rata - rata kompleksitas $O(n \log n)$ juga dari Faro (2020). Juga didukung oleh waktu run time dari masing - masing algoritma, kita menemukan bahwa Bubble sort memiliki waktu runtime paling lambat untuk ukuran data 7000 yaitu 4.612817 detik, disusul oleh Insertion sort dengan waktu 2.477953 detik, dan terakhir Quick sort dengan 0.021861 detik.

Meskipun sebagian besar hasil uji coba sesuai dengan asumsi runtime melalui kompleksitas algoritma, terdapat kejanggalan dimana Insertion sort dapat memotong waktu hingga 46% dari waktu run time Bubblesort meskipun memiliki kompleksitas waktu yang sama $O(n^2)$. Mengapa demikian? Karena Insertion sort mendapat keuntungan dari anggota array yang sudah tersortir sebelumnya oleh algoritma itu sendiri, atau oleh data input, sementara Bubble sort akan tetap melakukan proses sortir terhadap data yang sudah di sort maupun belum jika data tersebut belum di sort oleh algoritma sebelumnya.

Kesimpulan

Quick sort merupakan metode sorting paling optimal antara ketiga algoritma tersebut, di susul oleh Insertion sort dan Bubble sort

Referensi

Sonita, A., & Nurtaneo, F. (2016). ANALISIS PERBANDINGAN ALGORITMA BUBBLE SORT, MERGE SORT, DAN QUICK SORT DALAM PROSES PENGURUTAN KOMBINASI ANGKA DAN HURUF. Pseudocode, 2(2), 75–80. https://doi.org/10.33369/pseudocode.2.2.75-80

Retnoningsih Endang. 2018. Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort. STMIK Bina Insani.

Davina Azalia Tara, Ihza Ferdina, M. Stevanza Sylvester, Muhammad Fiqi Firmansyah, Muhammad Sulthonul Izza, Nur Widhya Astuti, Rafael Putra Amarta, Ridwan Fajariansyah. 2024. Analisis Kompleksitas Waktu Menggunakan Sorting Algorithm Pada Pengaplikasian Fitur Pengurutan Harga dari Terendah dan Tertinggi Di Shopee. Universitas Negeri Semarang.

Joseph JaJa. 1999. *A Perspective on Quicksort*. IEEE Computer, Vol. 32, No. 4, pp. 70-72.

C. A. R. Hoare. 1962. Quicksort. *The Computer Journal*, Volume 5, Issue 1, Pages 10–16. University of Oxford.

Faro, S., Marino, F. P., & Scafiti, S. (2020). *Fast-Insertion-Sort: a New Family of Efficient Variants of the Insertion-Sort Algorithm*. Dalam *Proceedings of the 16th International Conference on Current Trends in Theory and Practice of Informatics (SOFSEM 2020)*. CEUR Workshop Proceedings, Vol-2568, hlm. 43-54.

Alake, R. (2024, December 6). Bubble Sort Time Complexity and Algorithm Explained. Built In. https://builtin.com/data-science/bubble-sort-time-complexity

File CSV bisa diakses disini :

https://github.com/Mranomalist62/TubesStragalKelompok5/blob/main/data/restaurant_data.csv