TUGAS 4

disusun untuk memenuhi tugas mata kuliah
Struktur Data dan Algoritma D

oleh:

Rahmad Hidayatullah Tsunami

2308107010035

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SYIAH KUALA
2025

Pendahuluan

1. Latar Belakang

Dalam dunia informatika, algoritma sorting memegang peranan penting dalam mengatur dan mengelola data secara efisien, terutama dalam skala besar. Berbagai algoritma seperti Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort memiliki karakteristik performa yang berbeda-beda, baik dari sisi kecepatan eksekusi maupun penggunaan memori. Oleh karena itu, pemahaman mendalam tentang bagaimana masing-masing algoritma bekerja dan bagaimana performanya dalam berbagai kondisi menjadi kompetensi penting bagi mahasiswa informatika. Melalui tugas ini, mahasiswa diharapkan dapat mengimplementasikan berbagai metode sorting, menguji kinerjanya pada data berjumlah jutaan, serta melakukan analisis berdasarkan hasil pengujian.

Tugas ini tidak hanya melatih kemampuan teknis dalam pemrograman menggunakan bahasa C, tetapi juga mendorong keterampilan eksperimental dalam mengumpulkan dan menganalisis data performa algoritma. Dengan membangkitkan data acak dalam jumlah besar dan menguji algoritma terhadap variasi ukuran data, mahasiswa akan belajar mengevaluasi kompleksitas waktu dan ruang dari masing-masing metode. Hasil analisis akan disajikan dalam bentuk tabel, grafik, serta laporan terstruktur yang mencakup deskripsi algoritma, perbandingan hasil, dan kesimpulan. Melalui pendekatan ini, diharapkan mahasiswa dapat mengembangkan kemampuan analitis dan kritis terhadap efisiensi algoritma dalam konteks dunia nyata.

2. Tujuan

Tujuan dari tugas ini adalah untuk memahami dan menganalisis performa berbagai algoritma sorting terhadap dataset besar. Secara spesifik, tujuan dari eksperimen ini meliputi:

a) Menerapkan algoritma sorting pada skala data besar **–** Mengembangkan implementasi algoritma Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort untuk pengurutan data dalam jumlah jutaan.

b) Menganalisis pengaruh ukuran data terhadap kinerja algoritma – Melakukan pengujian dengan variasi ukuran dataset untuk melihat bagaimana kompleksitas algoritma mempengaruhi kecepatan dan konsumsi memori.

c) Menarik kesimpulan terkait keunggulan dan kelemahan masing-masing algoritma **–** Menyusun analisis komparatif untuk memahami situasi di mana suatu algoritma lebih optimal dibandingkan algoritma lainnya.

3. Deskripsi Algoritma Sorting

1) Bubble Sort

a) Deskripsi : Bubble Sort adalah algoritma pengurutan sederhana yang bekerja dengan membandingkan pasangan elemen yang berdekatan dan menukarnya jika urutannya salah. Proses ini diulang hingga tidak ada lagi pertukaran yang diperlukan, menandakan bahwa array telah terurut.

b) Kompleksitas Waktu : $O(n^2)$

c) Algoritma :

1) Mulai dari indeks pertama array.
2) Bandingkan elemen saat ini dengan elemen berikutnya.
3) Jika elemen saat ini lebih besar dari elemen berikutnya, tukar posisi keduanya.
4) Lanjutkan ke pasangan elemen berikutnya dan ulangi langkah 2–3 hingga akhir array.
5) Ulangi proses dari langkah 1 untuk seluruh array sebanyak n-1 kali, di mana n adalah jumlah elemen dalam array.

2) Selection Sort

a) Deskripsi : Selection Sort mengurutkan array dengan cara berulang kali mencari elemen terkecil dari bagian array yang belum terurut dan menukarnya dengan elemen pertama dari bagian tersebut. Proses ini diulang untuk setiap posisi dalam array hingga seluruh array terurut.

b) Kompleksitas Waktu : $O(n^2)$

c) Algoritma :

1) Mulai dari indeks pertama array sebagai posisi saat ini.
2) Cari elemen terkecil dalam subarray yang dimulai dari posisi saat ini hingga akhir array.
3) Tukar elemen terkecil yang ditemukan dengan elemen pada posisi saat ini.
4) Pindah ke indeks berikutnya dan ulangi langkah 2–3 hingga mencapai akhir array.

3) Insertion Sort

a) Deskripsi : Insertion Sort membangun array terurut satu per satu dengan mengambil satu elemen dari array yang belum terurut dan menyisipkannya ke posisi yang tepat dalam array yang sudah terurut. Proses ini mirip dengan cara seseorang menyusun kartu di tangan.

b) Kompleksitas Waktu : $O(n^2)$

c) **Algoritma :**
   1) Anggap elemen pertama sebagai array yang sudah terurut.
   2) Ambil elemen berikutnya sebagai elemen kunci.
   3) Bandingkan kunci dengan elemen-elemen dalam array terurut dari belakang ke depan.
   4) Geser elemen yang lebih besar dari kunci satu posisi ke kanan.
   5) Tempatkan kunci pada posisi yang sesuai.
   6) Ulangi langkah 2–5 untuk semua elemen dalam array.

4) Merge Sort

a) **Deskripsi :** Merge Sort adalah algoritma pengurutan yang menggunakan pendekatan divide and conquer. Algoritma ini membagi array menjadi dua bagian, mengurutkan setiap bagian secara rekursif, dan kemudian menggabungkan kedua bagian tersebut menjadi array yang terurut.

b) **Kompleksitas Waktu :** O(n log n)

c) **Algoritma :**
   1) Jika array memiliki lebih dari satu elemen:
      - Bagi array menjadi dua bagian tengah.
      - Rekursif lakukan Merge Sort pada setiap bagian.
   2) Setelah kedua bagian terurut, gabungkan keduanya menjadi satu array terurut dengan cara:
      - Bandingkan elemen pertama dari kedua bagian.
      - Tempatkan elemen yang lebih kecil ke array hasil.
      - Ulangi proses hingga semua elemen dari kedua bagian telah digabungkan.

5) Quick Sort

a) **Deskripsi :** Quick Sort adalah algoritma pengurutan yang menggunakan pendekatan divide and conquer. Algoritma ini memilih satu elemen sebagai pivot dan mempartisi array sehingga elemen yang lebih kecil dari pivot berada di sebelah kiri, dan elemen yang lebih besar berada di sebelah kanan. Proses ini diulang secara rekursif untuk subarray di kiri dan kanan pivot.

b) **Kompleksitas Waktu :** O(n log n) rata-rata, $O(n^2)$ dalam kasus terburuk

c) **Algoritma :**
   1) Jika array memiliki lebih dari satu elemen:
      - Pilih elemen pivot (misalnya, elemen terakhir).

- Partisi array menjadi dua subarray: Elemen yang lebih kecil dari pivot dan Elemen yang lebih besar dari pivot.
- Rekursif lakukan Quick Sort pada kedua subarray.

2) Gabungkan subarray yang telah terurut dan pivot menjadi satu array terurut.

6) Shell Sort

a) Deskripsi : Shell Sort adalah versi yang dioptimalkan dari Insertion Sort. Algoritma ini mengurutkan elemen yang terpisah dengan jarak tertentu (gap), yang secara bertahap dikurangi hingga menjadi 1, di mana array diurutkan menggunakan Insertion Sort. Pendekatan ini membantu memindahkan elemen ke posisi yang lebih dekat dengan posisi akhirnya lebih cepat.

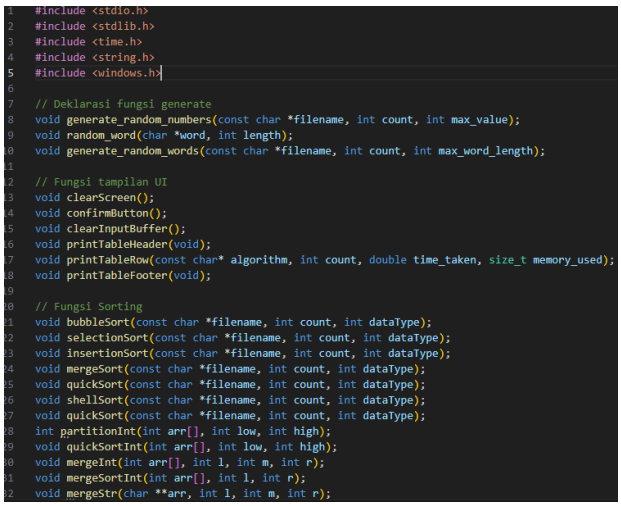
b) Kompleksitas Waktu : $O(n \log^2 n)$

c) Algoritma :

1) Tentukan nilai awal gap (misalnya, setengah dari panjang array).
2) Lakukan pengurutan elemen yang terpisah dengan jarak gap menggunakan Insertion Sort.
3) Kurangi nilai gap (misalnya, bagi dua).
4) Ulangi langkah 2–3 hingga gap menjadi 1.
5) Lakukan pengurutan akhir dengan gap 1 menggunakan Insertion Sort.

Implementasi Kode C

a) File header.h



```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <windows.h>

// Deklarasi fungsi generate
void generate_random_numbers(const char *filename, int count, int max_value);
void random_word(char *word, int length);
void generate_random_words(const char *filename, int count, int max_word_length);

// Fungsi tampilan UI
void clearScreen();
void confirmButton();
void clearInputBuffer();
void printTableHeader(void);
void printTableRow(const char* algorithm, int count, double time_taken, size_t memory_used);
void printTableFooter(void);

// Fungsi Sorting
void bubbleSort(const char *filename, int count, int dataType);
void selectionSort(const char *filename, int count, int dataType);
void insertionSort(const char *filename, int count, int dataType);
void mergeSort(const char *filename, int count, int dataType);
void quickSort(const char *filename, int count, int dataType);
void shellSort(const char *filename, int count, int dataType);
void quickSort(const char *filename, int count, int dataType);
int partitionInt(int arr[], int low, int high);
void quickSortInt(int arr[], int low, int high);
void mergeInt(int arr[], int l, int m, int r);
void mergeSortInt(int arr[], int l, int r);
void mergeStr(char **arr, int l, int m, int r);
```

File header.h berisi seluruh deklarasi fungsi dan sebagian besar implementasi untuk kebutuhan program, khususnya fungsi-fungsi berikut:

- **Fungsi Generate Data**
  Menyediakan fungsi generate_random_numbers() dan generate_random_words() untuk membuat file data berisi angka atau kata acak dalam jumlah besar, serta fungsi random_word() untuk membangkitkan kata acak.

- **Fungsi Tampilan (UI)**
  Menyediakan utilitas seperti clearScreen(), clearInputBuffer(), confirmButton(), serta fungsi untuk mencetak tabel hasil percobaan (printTableHeader(), printTableRow(), printTableFooter()).

- **Fungsi Sorting**
  Implementasi berbagai algoritma sorting yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Setiap algoritma mendukung pengurutan untuk data bertipe angka maupun kata.

- **Struktur Modular**
  File ini memisahkan fungsi berdasarkan kategori (generate, UI, sorting) dan membuatnya dapat digunakan kembali dari file .c lain seperti main.c.

- **Penggunaan Memory dan Timing**
  Tiap fungsi sorting mencatat waktu eksekusi menggunakan clock() dan menghitung penggunaan memori berdasarkan tipe data yang diproses.

b) File main.c

```c
#include "header.h"

/*— enumerasi ragam menu —*/
enum DataKind   { DATA_ANGKA = 1, DATA_HURUF };
enum DataLength {
    LEN_10K = 1, LEN_50K, LEN_100K, LEN_250K,
    LEN_500K, LEN_1M, LEN_1_5M, LEN_2M
};
enum SortMethod {
    SORT_BUBBLE = 1, SORT_SELECTION, SORT_INSERTION,
    SORT_MERGE, SORT_QUICK, SORT_SHELL, MENU_BACK
};

int main(void)
{
    enum DataKind dataOption;
    enum DataLength lengthOption;
    enum SortMethod methodOption;

    // cek data, jika tidak ada maka generate ulang
    FILE *fp = fopen("dataNumber.txt", "r");
    if (!fp) {
        puts("< File dataNumber.txt tidak ditemukan, membuat data baru... >");
        Sleep(1200);
        generate_random_numbers("dataNumber.txt", 200000000, 2000000); // contoh: 20 juta angka acak, max 2000000
    } else {
        fclose(fp); // file ditemukan, tutup kembali
    }

    fp = fopen("dataWord.txt", "r");
    if (!fp) {
        puts("< File dataWord.txt tidak ditemukan, membuat data baru... >");
```

File main.c bertindak sebagai program utama yang mengatur alur interaksi dengan pengguna:

- **Pengecekan dan Pembuatan File Data**
  Saat program dijalankan, akan diperiksa apakah file dataNumber.txt dan dataWord.txt sudah tersedia. Jika tidak, file akan dibuat secara otomatis menggunakan fungsi generate dari header.h.

- **Tampilan Menu Interaktif**
  Program menampilkan tiga menu utama kepada pengguna: memilih jenis data (angka atau kata), memilih ukuran dataset (10.000 hingga 2.000.000 data), dan memilih metode sorting yang diinginkan.

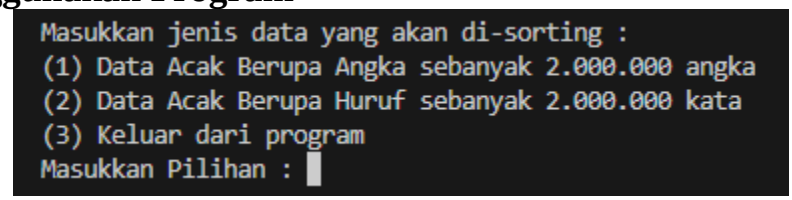
- **Konfirmasi Pengguna**
  Setelah pilihan dibuat, program meminta konfirmasi dari pengguna sebelum menjalankan proses sorting.

- **Pemanggilan Metode Sorting**
  Berdasarkan pilihan metode, main.c akan memanggil fungsi sorting yang sesuai (misalnya bubbleSort, mergeSort, dll.) dan menampilkan hasil waktu eksekusi dan penggunaan memori dalam format tabel.

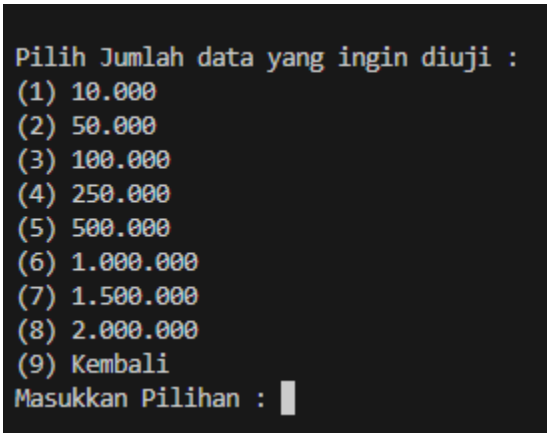
- **Opsi perbandingan Semua Method**
  Terdapat pilihan untuk menjalankan semua algoritma sorting sekaligus pada dataset yang sama, sehingga pengguna dapat membandingkan performa masing-masing metode dalam satu tampilan tabel.

Hasil Kode Program

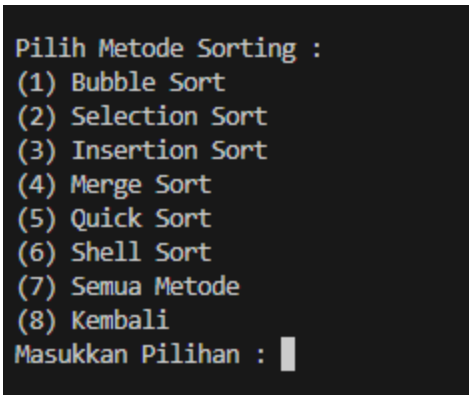
a) Cara menggunakan Program



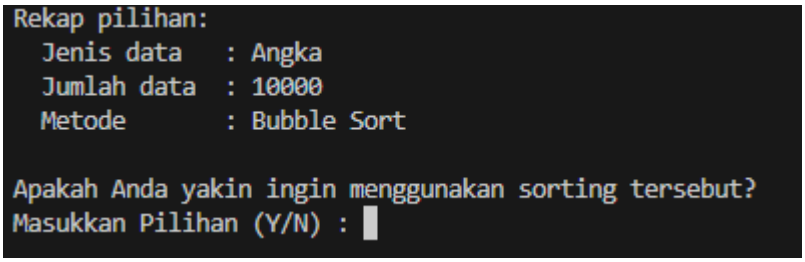
Pada bagian ini, user akan ditampilkan menu jenis data yang akan di sorting. User dapat memilih tipe data berupa angka (number) atau kata (string). Sebelum masuk ke menu pertama (menu ini), program akan menjalankan kode untuk mengecek file "dataNumber.txt" dan file "dataWords.txt" tersedia di direktori. kedua file ini berisi tentang kata dan angka acak yang terdiri dari 20 juta baris. Jika kedua file ini tidak tersedia, maka program akan menjalakn fungsi generate_random_numbers() dan generate_random_words() untuk membuat file tersebut.



Setelah memilih jenis data yang akan disorting muncul menu di atas. Pada menu ini, user akan memilih berapa banyak data yang akan disorting.

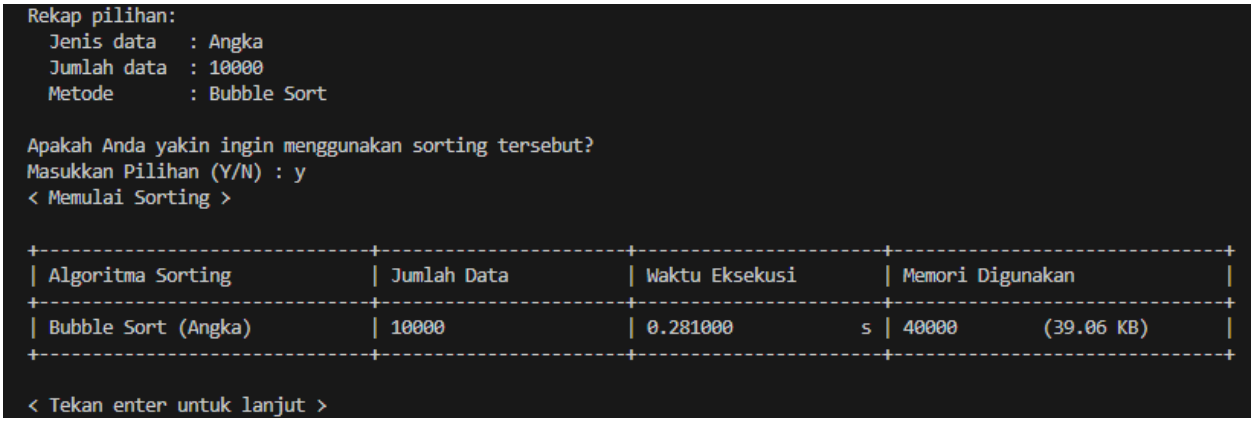


Setelah memilih jumlah data yang akan disorting, muncul menu di atas. Pada menu tersebut, user akan memilih metode sorting yang akan akan diimplementasikan pada data yang akan di sorting. User dapat memilih salah satu metode atau memilih semua metode sorting. Hasil data yang akan disorting akan ditampilakn dalam bentuk tabel.



Setelah memilih ketiga menu sebelumnya, user akan diminta konfirmasi pilihan yang telah dipilih. Jika ya, lanjut ke proses sorting dan tampilkan hasil sorting. Jika tidak, kembali ke menu awal.

Contoh hasil sorting :



b) Perbandingan semua metode sorting

- **Data Angka (Number)**

1) 10.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 10000                | 0.168000           s | 40000        (39.06 KB)      |
| Selection Sort (Angka)       | 10000                | 0.070000           s | 40000        (39.06 KB)      |
| Insertion Sort (Angka)       | 10000                | 0.048000           s | 40000        (39.06 KB)      |
| Merge Sort (Angka)           | 10000                | 0.001000           s | 40000        (39.06 KB)      |
| Quick Sort (Angka)           | 10000                | 0.000000           s | 40000        (39.06 KB)      |
| Shell Sort (Angka)           | 10000                | 0.000000           s | 40000        (39.06 KB)      |
+------------------------------+----------------------+----------------------+------------------------------+
```

2) 50.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 50000                | 5.662000           s | 200000       (195.31 KB)     |
| Selection Sort (Angka)       | 50000                | 1.809000           s | 200000       (195.31 KB)     |
| Insertion Sort (Angka)       | 50000                | 1.223000           s | 200000       (195.31 KB)     |
| Merge Sort (Angka)           | 50000                | 0.015000           s | 200000       (195.31 KB)     |
| Quick Sort (Angka)           | 50000                | 0.004000           s | 200000       (195.31 KB)     |
| Shell Sort (Angka)           | 50000                | 0.009000           s | 200000       (195.31 KB)     |
+------------------------------+----------------------+----------------------+------------------------------+
```

3) 100.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 100000               | 23.411000          s | 400000       (390.63 KB)     |
| Selection Sort (Angka)       | 100000               | 7.249000           s | 400000       (390.63 KB)     |
| Insertion Sort (Angka)       | 100000               | 4.858000           s | 400000       (390.63 KB)     |
| Merge Sort (Angka)           | 100000               | 0.029000           s | 400000       (390.63 KB)     |
| Quick Sort (Angka)           | 100000               | 0.011000           s | 400000       (390.63 KB)     |
| Shell Sort (Angka)           | 100000               | 0.029000           s | 400000       (390.63 KB)     |
+------------------------------+----------------------+----------------------+------------------------------+
```

4) 250.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 250000               | 147.541000         s | 1000000      (976.56 KB)     |
| Selection Sort (Angka)       | 250000               | 51.335000          s | 1000000      (976.56 KB)     |
| Insertion Sort (Angka)       | 250000               | 31.346000          s | 1000000      (976.56 KB)     |
| Merge Sort (Angka)           | 250000               | 0.073000           s | 1000000      (976.56 KB)     |
| Quick Sort (Angka)           | 250000               | 0.028000           s | 1000000      (976.56 KB)     |
| Shell Sort (Angka)           | 250000               | 0.045000           s | 1000000      (976.56 KB)     |
+------------------------------+----------------------+----------------------+------------------------------+
```

5) 500.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 500000               | 605.272000         s | 2000000      (1953.13 KB)    |
| Selection Sort (Angka)       | 500000               | 200.454000         s | 2000000      (1953.13 KB)    |
| Insertion Sort (Angka)       | 500000               | 127.895000         s | 2000000      (1953.13 KB)    |
| Merge Sort (Angka)           | 500000               | 0.148000           s | 2000000      (1953.13 KB)    |
| Quick Sort (Angka)           | 500000               | 0.070000           s | 2000000      (1953.13 KB)    |
| Shell Sort (Angka)           | 500000               | 0.122000           s | 2000000      (1953.13 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

6) 1.000.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 1000000              | 14628.889000       s | 24000000     (23437.50 KB)   |
| Selection Sort (Kata)        | 1000000              | 8703.784000        s | 24000000     (23437.50 KB)   |
| Insertion Sort (Kata)        | 1000000              | 7672.538000        s | 24000000     (23437.50 KB)   |
| Merge Sort (Kata)            | 1000000              | 0.487000           s | 24000000     (23437.50 KB)   |
| Quick Sort (Kata)            | 1000000              | 0.416000           s | 24000000     (23437.50 KB)   |
| Shell Sort (Kata)            | 1000000              | 1.688000           s | 24000000     (23437.50 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
```

7) 1.500.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 1500000              | 5509.940000        s | 6000000      (5859.38 KB)    |
| Selection Sort (Angka)       | 1500000              | 1821.081000        s | 6000000      (5859.38 KB)    |
| Insertion Sort (Angka)       | 1500000              | 1164.929000        s | 6000000      (5859.38 KB)    |
| Merge Sort (Angka)           | 1500000              | 0.438000           s | 6000000      (5859.38 KB)    |
| Quick Sort (Angka)           | 1500000              | 0.282000           s | 6000000      (5859.38 KB)    |
| Shell Sort (Angka)           | 1500000              | 0.427000           s | 6000000      (5859.38 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

8) 2.000.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Angka)          | 2000000              | 11777.717000       s | 8000000      (7812.50 KB)    |
| Selection Sort (Angka)       | 2000000              | 3445.325000        s | 8000000      (7812.50 KB)    |
| Insertion Sort (Angka)       | 2000000              | 2150.795000        s | 8000000      (7812.50 KB)    |
| Merge Sort (Angka)           | 2000000              | 0.693000           s | 8000000      (7812.50 KB)    |
| Quick Sort (Angka)           | 2000000              | 0.492000           s | 8000000      (7812.50 KB)    |
| Shell Sort (Angka)           | 2000000              | 0.672000           s | 8000000      (7812.50 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

- **Data Kata (String)**

1) 10.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 10000                | 0.386000           s | 240000       (234.38 KB)     |
| Selection Sort (Kata)        | 10000                | 0.149000           s | 240000       (234.38 KB)     |
| Insertion Sort (Kata)        | 10000                | 0.071000           s | 240000       (234.38 KB)     |
| Merge Sort (Kata)            | 10000                | 0.004000           s | 240000       (234.38 KB)     |
| Quick Sort (Kata)            | 10000                | 0.001000           s | 240000       (234.38 KB)     |
| Shell Sort (Kata)            | 10000                | 0.002000           s | 240000       (234.38 KB)     |
+------------------------------+----------------------+----------------------+------------------------------+
```

2) 50.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 50000                | 11.948000          s | 1200000      (1171.88 KB)    |
| Selection Sort (Kata)        | 50000                | 4.385000           s | 1200000      (1171.88 KB)    |
| Insertion Sort (Kata)        | 50000                | 2.180000           s | 1200000      (1171.88 KB)    |
| Merge Sort (Kata)            | 50000                | 0.017000           s | 1200000      (1171.88 KB)    |
| Quick Sort (Kata)            | 50000                | 0.002000           s | 1200000      (1171.88 KB)    |
| Shell Sort (Kata)            | 50000                | 0.016000           s | 1200000      (1171.88 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

3) 100.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 100000               | 59.264000          s | 2400000      (2343.75 KB)    |
| Selection Sort (Kata)        | 100000               | 21.850000          s | 2400000      (2343.75 KB)    |
| Insertion Sort (Kata)        | 100000               | 9.614000           s | 2400000      (2343.75 KB)    |
| Merge Sort (Kata)            | 100000               | 0.037000           s | 2400000      (2343.75 KB)    |
| Quick Sort (Kata)            | 100000               | 0.020000           s | 2400000      (2343.75 KB)    |
| Shell Sort (Kata)            | 100000               | 0.052000           s | 2400000      (2343.75 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

4) 250.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 250000               | 619.460000         s | 6000000      (5859.38 KB)    |
| Selection Sort (Kata)        | 250000               | 337.981000         s | 6000000      (5859.38 KB)    |
| Insertion Sort (Kata)        | 250000               | 236.335000         s | 6000000      (5859.38 KB)    |
| Merge Sort (Kata)            | 250000               | 0.105000           s | 6000000      (5859.38 KB)    |
| Quick Sort (Kata)            | 250000               | 0.067000           s | 6000000      (5859.38 KB)    |
| Shell Sort (Kata)            | 250000               | 0.238000           s | 6000000      (5859.38 KB)    |
+------------------------------+----------------------+----------------------+------------------------------+
```

5) 500.000 baris

```
=== Perbandingan Metode Sorting ===

+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Bubble Sort (Kata)           | 500000               | 5326.580000        s | 12000000     (11718.75 KB)   |
| Selection Sort (Kata)        | 500000               | 1770.057000        s | 12000000     (11718.75 KB)   |
| Insertion Sort (Kata)        | 500000               | 1299.758000        s | 12000000     (11718.75 KB)   |
| Merge Sort (Kata)            | 500000               | 0.259000           s | 12000000     (11718.75 KB)   |
| Quick Sort (Kata)            | 500000               | 0.181000           s | 12000000     (11718.75 KB)   |
| Shell Sort (Kata)            | 500000               | 0.760000           s | 12000000     (11718.75 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
```

6) 1.000.000 baris

```
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Merge Sort (Kata)            | 1000000              | 0.848000           s | 24000000     (23437.50 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Quick Sort (Kata)            | 1000000              | 0.713000           s | 24000000     (23437.50 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Shell Sort (Kata)            | 1000000              | 2.849000           s | 24000000     (23437.50 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
```

7) 1.500.000 baris

```
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Merge Sort (Kata)            | 1500000              | 1.362000           s | 36000000     (35156.25 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Quick Sort (Kata)            | 1500000              | 1.004000           s | 36000000     (35156.25 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
+------------------------------+----------------------+----------------------+------------------------------+
| Algoritma Sorting            | Jumlah Data          | Waktu Eksekusi       | Memori Digunakan             |
+------------------------------+----------------------+----------------------+------------------------------+
| Shell Sort (Kata)            | 1500000              | 4.282000           s | 36000000     (35156.25 KB)   |
+------------------------------+----------------------+----------------------+------------------------------+
```

8) **2.000.000 baris**

Algoritma Sorting	Jumlah Data	Waktu Eksekusi	Memori Digunakan
Merge Sort (Kata)	2000000	1.929000 s	48000000 (46875.00 KB)

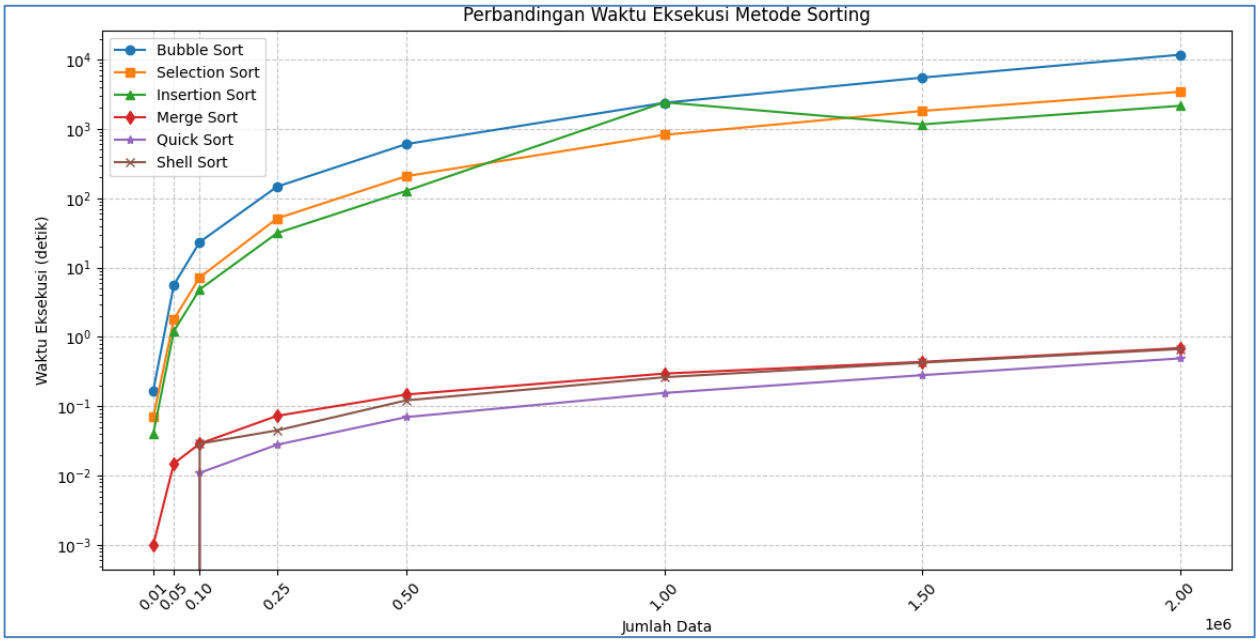
Algoritma Sorting	Jumlah Data	Waktu Eksekusi	Memori Digunakan
Quick Sort (Kata)	2000000	1.460000 s	48000000 (46875.00 KB)

Algoritma Sorting	Jumlah Data	Waktu Eksekusi	Memori Digunakan
Shell Sort (Kata)	2000000	6.217000 s	48000000 (46875.00 KB)

Untuk data kata (string) yang lebih dari 1.000.000, metode bubble sort, selection sort, dan insertion sort dibutuhkan waktu yang sangat lama. Jadi, untuk bagian ini hanya menampilkan metode merge sort, quick sort dan shell sort saja.

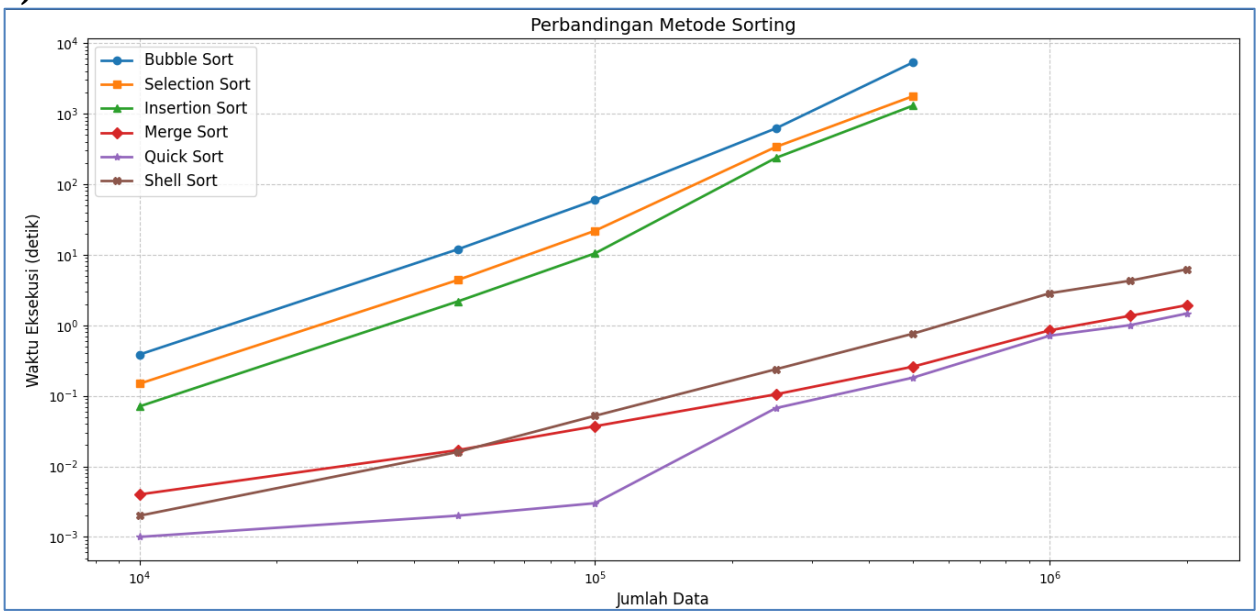
C) Visualisasi perbandingan metode sorting

1) Data Angka



Berdasarkan visualisasi tersebut, terlihat jelas perbedaan kinerja yang signifikan antara algoritma sorting. Bubble Sort menunjukkan peningkatan waktu eksekusi yang paling drastis seiring bertambahnya jumlah data, mencapai lebih dari 10.000 detik pada 2 juta data, membuktikan kompleksitas $O(n^2)$ yang tidak efisien untuk dataset besar. Selection Sort dan Insertion Sort juga menunjukkan pola pertumbuhan eksponensial yang serupa meskipun sedikit lebih baik dari Bubble Sort, dengan waktu eksekusi mencapai ribuan detik pada data berjumlah besar. Sebaliknya, tiga algoritma lainnya yaitu, Merge Sort, Quick Sort, dan Shell Sort memperlihatkan performa yang jauh lebih efisien dengan waktu eksekusi yang tetap rendah (kurang dari 1 detik) bahkan untuk 2 juta data, dengan Quick Sort sedikit lebih cepat, diikuti oleh Shell Sort dan Merge Sort, membuktikan keunggulan algoritma dengan kompleksitas $O(n \log n)$ untuk pengurutan data dalam jumlah besar.

2) Data Kata



Visualisasi grafik tersebut menunjukkan perbandingan kinerja enam algoritma sorting dengan skala logaritmik, di mana terlihat jelas perbedaan efisiensi yang signifikan antara algoritma $O(n^2)$ dan $O(n \log n)$. Bubble Sort konsisten menunjukkan performa terburuk dengan waktu eksekusi meningkat drastis hingga mencapai ribuan detik pada 500.000 data, diikuti oleh Selection Sort dan Insertion Sort yang juga menunjukkan pertumbuhan waktu yang eksponensial. Sebaliknya, ketiga algoritma efisien (Quick Sort, Merge Sort, dan Shell Sort) mempertahankan kinerja yang jauh lebih baik bahkan pada dataset besar, dengan Quick Sort tampil sebagai algoritma tercepat dengan waktu eksekusi hanya sekitar 1,5 detik untuk 2 juta data, diikuti oleh Merge Sort, sementara Shell Sort meski lebih cepat dari algoritma $O(n^2)$, tetap membutuhkan waktu sekitar 6 detik untuk jumlah data yang sama, mendemonstrasikan dengan jelas mengapa algoritma $O(n \log n)$ sangat direkomendasikan untuk pengurutan data dalam jumlah besar.

Kesimpulan :

Berdasarkan analisis perbandingan kinerja enam algoritma sorting pada kedua dataset (angka dan string), terlihat pola konsisten di mana algoritma dengan kompleksitas $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu eksekusi yang drastis dan tidak praktis untuk dataset besar, dengan Bubble Sort selalu menempati posisi terburuk. Sebaliknya, algoritma dengan kompleksitas O(n log n) yaitu Quick Sort, Merge Sort, dan Shell Sort memperlihatkan efisiensi yang jauh lebih tinggi bahkan pada dataset berjumlah jutaan, dengan Quick Sort secara konsisten menjadi yang tercepat (sekitar 1,5 detik untuk 2 juta data), diikuti oleh Merge Sort, sedangkan Shell Sort meskipun lebih baik dari algoritma $O(n^2)$ tetapi membutuhkan waktu lebih lama (sekitar 6 detik untuk 2 juta data) dibandingkan dua algoritma O(n log n) lainnya, sehingga jelas bahwa untuk pengolahan data dalam jumlah besar, algoritma O(n log n) terutama Quick Sort menjadi pilihan yang paling direkomendasikan karena efisiensinya yang superior.