

TUGAS 4

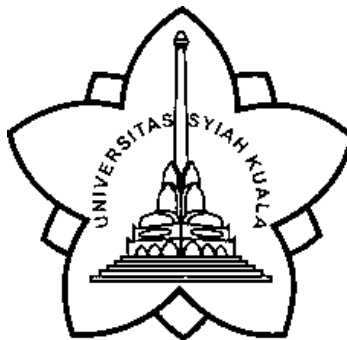
disusun untuk memenuhi

Tugas mata kuliah Struktur Data & Algoritma D

Oleh:

MUHAMMAD SYUKRI

2308107010060



JURUSAN INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

PENDAHULUAN

Sorting atau pengurutan data merupakan salah satu proses fundamental dalam ilmu komputer yang berperan penting dalam berbagai aplikasi, seperti pencarian data, pengolahan informasi, hingga optimasi algoritma. Beragam metode sorting telah dikembangkan dengan kompleksitas dan karakteristik yang berbeda-beda, mulai dari yang sederhana hingga yang efisien untuk skala besar. Dalam eksperimen ini, dilakukan pengujian terhadap enam algoritma sorting klasik, yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Tujuan dari eksperimen ini adalah untuk mengevaluasi performa masing-masing algoritma berdasarkan waktu eksekusi dan penggunaan memori dalam mengurutkan data berupa string (kata). Hasil dari eksperimen ini diharapkan dapat memberikan gambaran mengenai keunggulan dan kelemahan dari setiap algoritma, serta memberikan pertimbangan dalam memilih algoritma yang tepat sesuai kebutuhan.

DESKRIPSI ALGORITMA DAN CARA IMPLEMENTASI

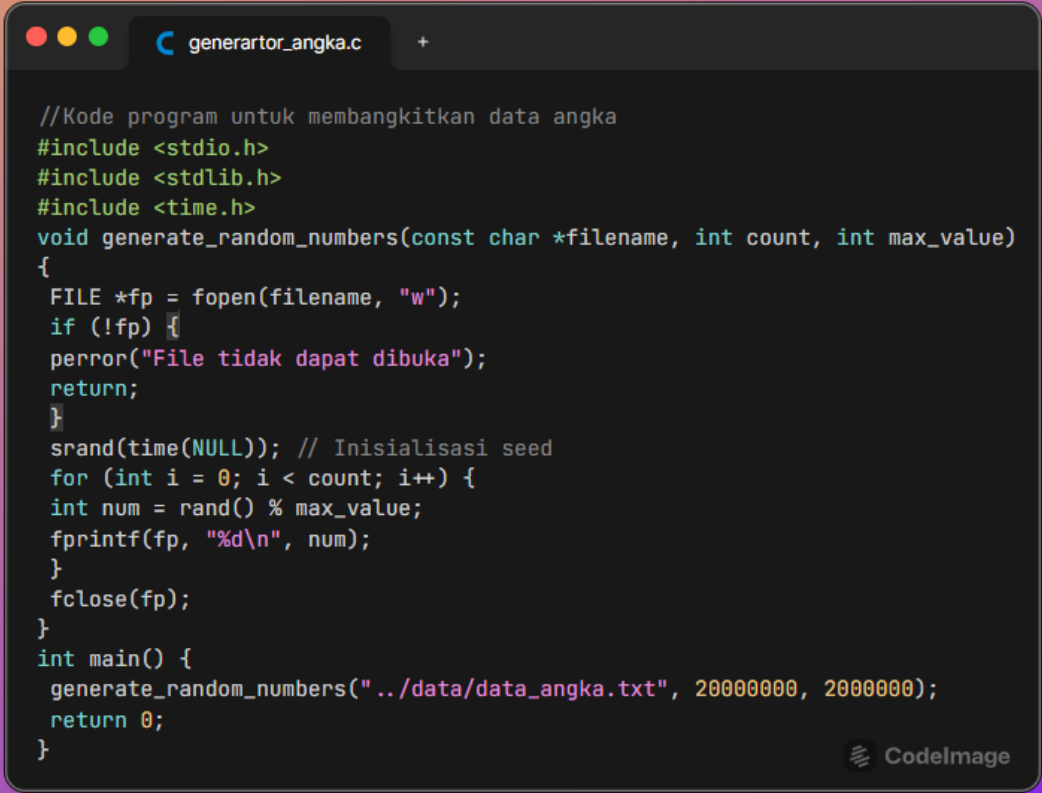
1. Bubble Sort membandingkan elemen berdekatan dan menukarnya jika salah urut, mudah dipahami tapi tidak efisien untuk data besar.
2. Selection Sort memilih elemen terkecil dan menempatkannya di posisi yang tepat, stabil dalam penggunaan memori, namun lambat.
3. Insertion Sort menyisipkan elemen ke posisi yang tepat dalam array terurut, efisien untuk data hampir terurut, namun kurang optimal untuk data acak besar.
4. Merge Sort menggunakan pendekatan *divide and conquer*, membagi array menjadi dua, mengurutkannya, lalu menggabungkannya kembali.
5. Quick Sort juga *divide and conquer*, memilih pivot dan membagi array berdasarkan nilai pivot, efisien untuk data acak.
6. Shell Sort adalah modifikasi Insertion Sort, membandingkan elemen dengan jarak bertahap, lebih cepat dari Insertion Sort.

TAHAPAN IMPLEMENTASI

1. Membuat Generator data

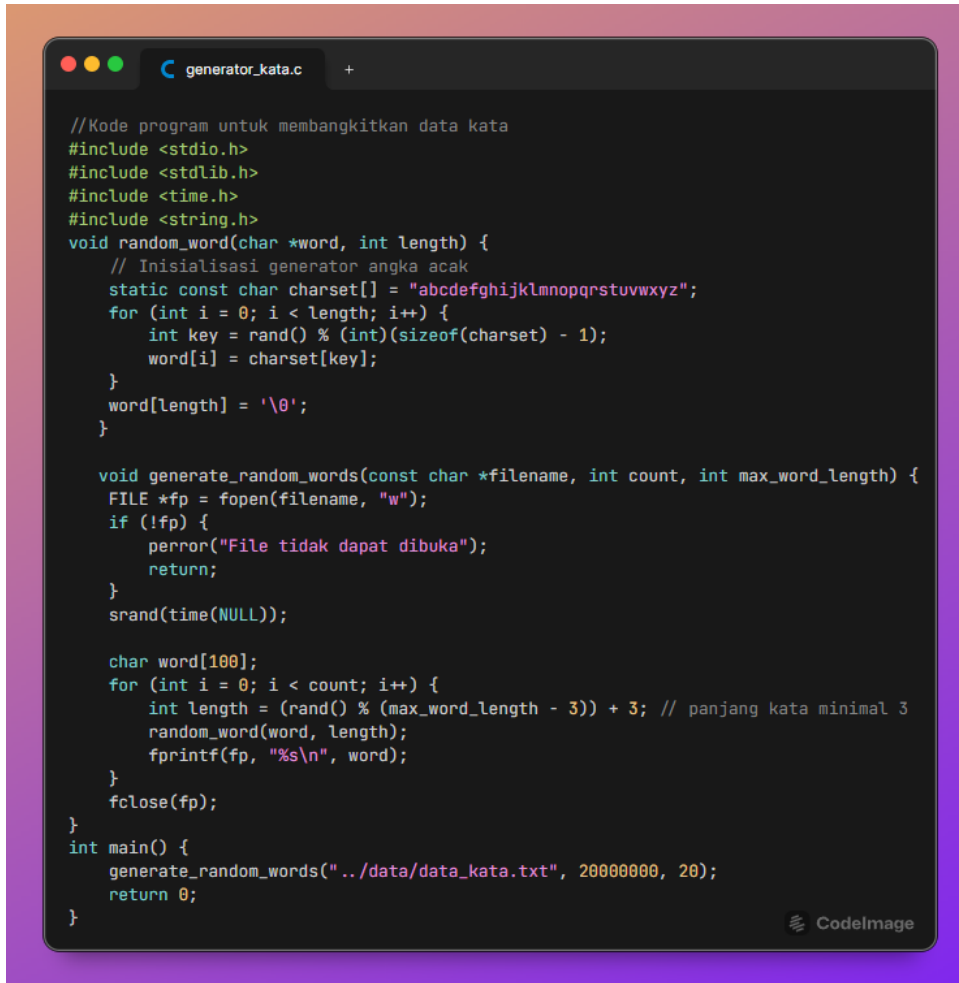
- Buat file generator_angka.c dan generator_kata.c untuk menghasilkan data acak sebanyak 20 juta angka dan kata.
- Simpan angka ke dalam folder data, tepatnya pada file file_angka.txt dan kata ke dalam file file_kata.txt.

2. Membuat File Kode Program Sorting



```
//Kode program untuk membangkitkan data angka
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
void generate_random_numbers(const char *filename, int count, int max_value)
{
    FILE *fp = fopen(filename, "w");
    if (!fp) {
        perror("File tidak dapat dibuka");
        return;
    }
    srand(time(NULL)); // Inisialisasi seed
    for (int i = 0; i < count; i++) {
        int num = rand() % max_value;
        fprintf(fp, "%d\n", num);
    }
    fclose(fp);
}
int main() {
    generate_random_numbers("../data/data_angka.txt", 20000000, 2000000);
    return 0;
}
```

Gambar 1. *Source code* untuk generate angka sebanyak 20 juta



```
//Kode program untuk membangkitkan data kata
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
void random_word(char *word, int length) {
    // Inisialisasi generator angka acak
    static const char charset[] = "abcdefghijklmnopqrstuvwxyz";
    for (int i = 0; i < length; i++) {
        int key = rand() % (int)(sizeof(charset) - 1);
        word[i] = charset[key];
    }
    word[length] = '\0';
}

void generate_random_words(const char *filename, int count, int max_word_length) {
    FILE *fp = fopen(filename, "w");
    if (!fp) {
        perror("File tidak dapat dibuka");
        return;
    }
    srand(time(NULL));

    char word[100];
    for (int i = 0; i < count; i++) {
        int length = (rand() % (max_word_length - 3)) + 3; // panjang kata minimal 3
        random_word(word, length);
        fprintf(fp, "%s\n", word);
    }
    fclose(fp);
}

int main() {
    generate_random_words("../data/data_kata.txt", 20000000, 20);
    return 0;
}
```

Gambar 2. *Source code* untuk generate kata sebanyak 20 juta

- Buat file sort.h yang menyimpan fungsi-fungsi sorting dan main.c yang akan mengeksekusi sorting sesuai dengan banyaknya data.

```

// =====
// Selection Sort
// =====

// Angka
void selectionSortAngka(int arr[], int n) {
    int minIdx;
    for (int i = 0; i < n-1; i++) {
        minIdx = i;
        for (int j = i+1; j < n; j++) {
            if (arr[j] < arr[minIdx]) {
                minIdx = j;
            }
        }
        temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }
}

// Kata
void selectionSortKata(char* arr[], int n) {
    int minIdx;
    char* temp;
    for (int i = 0; i < n-1; i++) {
        minIdx = i;
        for (int j = i+1; j < n; j++) {
            if (strcmp(arr[j], arr[minIdx]) < 0) {
                minIdx = j;
            }
        }
        temp = arr[minIdx];
        arr[minIdx] = arr[i];
        arr[i] = temp;
    }
}

// =====
// Insertion Sort
// =====

// Angka
void insertionSortAngka(int arr[], int n) {
    int key;
    for (int i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j+1] = key;
    }
}

// Kata
void insertionSortKata(char* arr[], int n) {
    char* key;
    int j;
    for (int i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;
        while (j >= 0 && strcmp(arr[j], key) > 0) {
            arr[j+1] = arr[j];
            j = j - 1;
        }
        arr[j+1] = key;
    }
}

```

```

// =====
// Quick Sort
// =====

// Angka
int partitionAngka(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// Kata
int partitionKata(char* arr[], int low, int high) {
    char* pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (strcmp(arr[j], pivot) < 0) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// =====
// Merge Sort
// =====

// Angka
void mergeAngka(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int* a1 = malloc(n1 * sizeof(int));
    int* a2 = malloc(n2 * sizeof(int));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (a1[i] < a2[j]) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

// Kata
void mergeKata(char* arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    char* a1 = malloc(n1 * sizeof(char*));
    char* a2 = malloc(n2 * sizeof(char*));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort kata");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (strcmp(a1[i], a2[j]) < 0) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

```

```

// =====
// Quick Sort
// =====

// Angka
int partitionAngka(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// Kata
int partitionKata(char* arr[], int low, int high) {
    char* pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (strcmp(arr[j], pivot) < 0) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// =====
// Merge Sort
// =====

// Angka
void mergeAngka(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int* a1 = malloc(n1 * sizeof(int));
    int* a2 = malloc(n2 * sizeof(int));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (a1[i] < a2[j]) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

// Kata
void mergeKata(char* arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    char* a1 = malloc(n1 * sizeof(char*));
    char* a2 = malloc(n2 * sizeof(char*));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort kata");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (strcmp(a1[i], a2[j]) < 0) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

```

```

// =====
// Quick Sort
// =====

// Angka
int partitionAngka(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// Kata
int partitionKata(char* arr[], int low, int high) {
    char* pivot = arr[high];
    int i = (low - 1);
    for (int j = low; j <= high - 1; j++) {
        if (strcmp(arr[j], pivot) < 0) {
            i++;
            temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    temp = arr[i+1];
    arr[i+1] = arr[high];
    arr[high] = temp;
    return (i + 1);
}

// =====
// Merge Sort
// =====

// Angka
void mergeAngka(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int* a1 = malloc(n1 * sizeof(int));
    int* a2 = malloc(n2 * sizeof(int));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (a1[i] < a2[j]) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

// Kata
void mergeKata(char* arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    char* a1 = malloc(n1 * sizeof(char*));
    char* a2 = malloc(n2 * sizeof(char*));
    if (!a1 || !a2) {
        perror("Gagal mengalokasikan memori untuk merge sort kata");
        exit(EXIT_FAILURE);
    }
    for (i = 0; i < n1; i++) a1[i] = arr[l + i];
    for (j = 0; j < n2; j++) a2[j] = arr[m + 1 + j];
    i = j = 0; k = l;
    while (i < n1 && j < n2) {
        if (strcmp(a1[i], a2[j]) < 0) a[k++] = a1[i++];
        else a[k++] = a2[j++];
    }
    while (i < n1) a[k++] = a1[i++];
    while (j < n2) a[k++] = a2[j++];
    free(a1);
    free(a2);
}

```

```

        i = j = 0; k = l;
        while (i < n1 && j < n2) {
            if (strcmp(L[i], R[j]) <= 0)
                arr[k++] = L[i++];
            else
                arr[k++] = R[j++];
        }
        while (i < n1) arr[k++] = L[i++];
        while (j < n2) arr[k++] = R[j++];

        free(L);
        free(R);
    }

void mergeSortKata(char* arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSortKata(arr, l, m);
        mergeSortKata(arr, m + 1, r);
        mergeKata(arr, l, m, r);
    }
}

// =====
// Shell Sort
// =====

// Angka
void shellSortAngka(int arr[], int n) {
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

// Kata
void shellSortKata(char* arr[], int n) {
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            char* temp = arr[i];
            int j;
            for (j = i; j >= gap && strcmp(arr[j - gap], temp) > 0; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}

#endif // SORT_H

```

Gambar 3. *Source Code sort.h*

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "sort.h"

#define MAX_DATA 2500000 // Menambah ukuran maksimum untuk aman
#define MAX_WORD_LEN 50

// Fungsi Load Data
void load_numbers(const char *path, int array[], int amount) {
    FILE *file = fopen(path, "r");
    if (!file) {
        perror("Gagal membuka file angka");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < amount; i++) {
        if (fscanf(file, "%d", &array[i]) != 1) break;
    }
    fclose(file);
}

void load_words(const char *path, char *array[], int amount) {
    FILE *file = fopen(path, "r");
    if (!file) {
        perror("Gagal membuka file kata");
        exit(EXIT_FAILURE);
    }
    for (int i = 0; i < amount; i++) {
        array[i] = malloc(MAX_WORD_LEN * sizeof(char));
        if (!array[i]) {
            perror("Gagal mengalokasikan memori untuk kata");
            exit(EXIT_FAILURE);
        }
        if (fscanf(file, "%s", array[i]) != 1) {
            free(array[i]); // Bebaskan memori jika gagal membaca
            array[i] = NULL;
            break;
        }
    }
    fclose(file);
}

// Fungsi Copy Array
void copy_array(int destination[], int source[], int n) {
    memcpy(destination, source, n * sizeof(int)); // Lebih efisien menggunakan memcpy
}

void copy_word_array(char *destination[], char *source[], int n) {
    for (int i = 0; i < n; i++) {
        // Salin string, bukan pointer
        strcpy(destination[i], source[i]);
    }
}

// Pilihan Tipe Data
int choose_data_type() {
    int choice;
    printf("Pilih Tipe Data mm:\n");
    printf("1. Angka\n2. Kata\nPilih Anda [1-2]: ");
    scanf("%d", &choice);
    if (choice == 1 && choice == 2) {
        printf("Pilihan tidak valid.\n");
        exit(EXIT_FAILURE);
    }
    return choice;
}

// Header & Footer
void print_header() {
    printf("===== Pengumuman =====\n");
    printf("Algoritma yang Digunakan | Waktu yang Dibutuhkan (Detik) | Memori yang Digunakan (MB) | \n");
    printf("-----|-----|-----| \n");
}

void print_footer() {
    printf("\n");
}

// Wrapper Fungsi untuk Menyamakan Tipe mm
// Angka
void wrapperMergeSortAngka(int arr[], int n) {
    // Alokasi array tambahan untuk merge sort untuk mencegah rekursi yang terlalu dalam
    int temp = malloc(n * sizeof(int));
    if (!temp) {
        perror("Gagal mengalokasikan memori untuk merge sort angka");
        exit(EXIT_FAILURE);
    }
    mergeSortAngka(arr, 0, n - 1);
    free(temp);
}

void wrapperQuickSortAngka(int arr[], int n) {
    quickSortAngka(arr, 0, n - 1);
}

// Kata
void wrapperMergeSortKata(char *arr[], int n) {
    mergeSortKata(arr, 0, n - 1);
}

void wrapperQuickSortKata(char *arr[], int n) {
    quickSortKata(arr, 0, n - 1);
}

// Fungsi Utama
int main() {
    int data_type = choose_data_type();
    int data_size;
    printf("Masukkan jumlah data yang ingin diurutkan: ");
    scanf("%d", &data_size);

    if (data_size <= 0 || data_size > MAX_DATA) {
        printf("Jumlah data tidak valid. Masukan nilai antara 1 dan %d.\n", MAX_DATA);
        return 1;
    }

    printf("Melakukan pengurutan untuk %d data...\n", data_size);
    print_header();

    clock_t start_time, end_time;
    double duration;

    if (data_type == 1) {
        // Alokasi memori untuk data awal
        int *initial_data = malloc(sizeof(int) * data_size);
        if (!initial_data) {
            perror("Gagal dalam alokasi memori untuk data awal.\n");
            return 1;
        }

        // Alokasi memori untuk data yang akan diurutkan
        int *data = malloc(sizeof(int) * data_size);
        if (!data) {
            perror("Gagal dalam alokasi memori untuk data sorting.\n");
            free(initial_data);
            return 1;
        }

        load_numbers("../data/data_angka.txt", initial_data, data_size);

        const char *sort_names[] = {
            "Bubble Sort (angka)", "Selection Sort (angka)", "Insertion Sort (angka)",
            "Merge Sort (angka)", "Quick Sort (angka)", "Shell Sort (angka)"
        };
        void (*sort_functions[]) (int[], int) = {
            bubbleSortAngka, selectionSortAngka, insertionSortAngka,
            wrapperMergeSortAngka, wrapperQuickSortAngka, shellSortAngka
        };

        for (int i = 0; i < 6; i++) {
            copy_array(data, initial_data, data_size);
            start_time = clock();
            sort_functions[i](data, data_size);
            end_time = clock();
            duration = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
            printf(" %-37s %-30.3f %-31.2f \n", sort_names[i], duration, sizeof(int) * data_size / (1024.0 * 1024.0));
        }

        free(initial_data);
        free(data);
    } else {
        // Alokasi array untuk pointer kata
        char *words = malloc(sizeof(char) * data_size);
        if (!words) {
            perror("Gagal dalam alokasi memori untuk array kata.\n");
            return 1;
        }
    }
}
```

```
// Alokasi memori untuk setiap kata di array copy
for (int i = 0; i < data_size; i++) {
    words_copy[i] = malloc(MAX_WORD_LEN * sizeof(char));
    if (!words_copy[i]) {
        // Bebaskan semua memori yang sudah dialokasikan jika gagal
        for (int j = 0; j < i; j++) {
            free(words_copy[j]);
        }
        free(words_copy);
        free(words);
        printf("Gagal dalam alokasi memori untuk kata copy.\n");
        return 1;
    }
}

load_words("../data/data_kata.txt", words, data_size);

const char *sort_names[] = {
    "Bubble Sort (kata)", "Selection Sort (kata)", "Insertion Sort (kata)",
    "Merge Sort (kata)", "Quick Sort (kata)", "Shell Sort (kata)"
};
void (*sort_functions[]) (char*[], int) = {
    bubbleSortKata, selectionSortKata, insertionSortKata,
    wrapperMergeSortKata, wrapperQuickSortKata, shellSortKata
};

for (int i = 0; i < 6; i++) {
    // Copy data kata
    copy_word_array(words_copy, words, data_size);

    start_time = clock();
    sort_functions[i](words_copy, data_size);
    end_time = clock();

    duration = ((double)(end_time - start_time)) / CLOCKS_PER_SEC;
    printf(" %-37s %-30.3f %-31.2f \n", sort_names[i], duration, sizeof(char) * data_size * MAX_WORD_LEN / (1024.0 * 1024.0));
}

// Pebebasan memori
for (int i = 0; i < data_size; i++) {
    free(words[i]);
    free(words_copy[i]);
}
free(words);
free(words_copy);

print_footer();
return 0;
}
```

Gambar 4. Source code main.c

HASIL PENGUJIAN

- DATA ANGKA

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1

Menjalankan pengurutan untuk 10000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	0.175	0.04
Selection Sort (angka)	0.083	0.04
Insertion Sort (angka)	0.063	0.04
Merge Sort (angka)	0.001	0.04
Quick Sort (angka)	0.001	0.04
Shell Sort (angka)	0.002	0.04

Gambar 5. Pengujian data angka sebanyak 10000 data

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1

Menjalankan pengurutan untuk 50000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	7.302	0.19
Selection Sort (angka)	1.802	0.19
Insertion Sort (angka)	1.227	0.19
Merge Sort (angka)	0.008	0.19
Quick Sort (angka)	0.005	0.19
Shell Sort (angka)	0.012	0.19

Gambar 6. Pengujian data angka sebanyak 50000 data

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1

Menjalankan pengurutan untuk 100000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	25.856	0.38
Selection Sort (angka)	6.351	0.38
Insertion Sort (angka)	4.861	0.38
Merge Sort (angka)	0.019	0.38
Quick Sort (angka)	0.019	0.38
Shell Sort (angka)	0.024	0.38

Gambar 7. Pengujian data angka sebanyak 100000 data


```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1

Menjalankan pengurutan untuk 250000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	154.343	0.95
Selection Sort (angka)	40.895	0.95
Insertion Sort (angka)	30.631	0.95
Merge Sort (angka)	0.043	0.95
Quick Sort (angka)	0.028	0.95
Shell Sort (angka)	0.057	0.95

Gambar 8. Pengujian data angka sebanyak 250000 data

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1

Menjalankan pengurutan untuk 500000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	611.334	1.91
Selection Sort (angka)	160.071	1.91
Insertion Sort (angka)	122.317	1.91
Merge Sort (angka)	0.078	1.91
Quick Sort (angka)	0.061	1.91
Shell Sort (angka)	0.116	1.91

Gambar 9. Pengujian data angka sebanyak 500000 data

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1
Masukkan jumlah data yang ingin diurutkan: 1000000

Menjalankan pengurutan untuk 1000000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	1887.601	3.81
Selection Sort (angka)	475.842	3.81
Insertion Sort (angka)	377.855	3.81
Merge Sort (angka)	0.215	3.81
Quick Sort (angka)	0.089	3.81
Shell Sort (angka)	0.212	3.81

Gambar 10. Pengujian data angka sebanyak 1000000 data

```
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1
Masukkan jumlah data yang ingin diurutkan: 1500000

Menjalankan pengurutan untuk 1500000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	4245.75	12.62
Selection Sort (angka)	1069.69	12.62
Insertion Sort (angka)	848.25	12.62
Merge Sort (angka)	0.425	12.62
Quick Sort (angka)	0.220	12.62
Shell Sort (angka)	0.565	12.62

Gambar 11. Pengujian data angka sebanyak 1500000 data

```
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 1
Masukkan jumlah data yang ingin diurutkan: 2000000

Menjalankan pengurutan untuk 2000000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (angka)	7548.45	16.1
Selection Sort (angka)	1897.103	16.1
Insertion Sort (angka)	1508.80	16.1
Merge Sort (angka)	0.451	16.1
Quick Sort (angka)	0.187	16.1
Shell Sort (angka)	0.453	16.1

Gambar 12. Pengujian data angka sebanyak 2000000 data

- DATA KATA

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> gcc main.c sort.h -o sort
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2

Menjalankan pengurutan untuk 10000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	0.883	0.48
Selection Sort (kata)	0.120	0.48
Insertion Sort (kata)	0.000	0.48
Merge Sort (kata)	0.009	0.48
Quick Sort (kata)	1.021	0.48
Shell Sort (kata)	0.000	0.48

Gambar 13. Pengujian data kata sebanyak 10000 data

```

PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> gcc main.c sort.h -o sort
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 50000

Menjalankan pengurutan untuk 50000 data...

```

Hasil Pengurutan:

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	10.843	2.38
Selection Sort (kata)	4.564	2.38
Insertion Sort (kata)	2.548	2.38
Merge Sort (kata)	0.016	2.38
Quick Sort (kata)	0.008	2.38
Shell Sort (kata)	0.023	2.38

Gambar 14. Pengujian data kata sebanyak 50000 data

```

PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> gcc main.c sort.h -o sort
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 100000

Menjalankan pengurutan untuk 100000 data...

```

Hasil Pengurutan:

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	44.759	4.77
Selection Sort (kata)	19.454	4.77
Insertion Sort (kata)	10.412	4.77
Merge Sort (kata)	0.043	4.77
Quick Sort (kata)	0.022	4.77
Shell Sort (kata)	0.048	4.77

Gambar 15. Pengujian data kata sebanyak 100000 data

```

PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> gcc main.c sort.h -o sort
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 250000

Menjalankan pengurutan untuk 250000 data...

```

Hasil Pengurutan:

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	739.448	11.92
Selection Sort (kata)	551.504	11.92
Insertion Sort (kata)	281.114	11.92
Merge Sort (kata)	0.145	11.92
Quick Sort (kata)	0.094	11.92
Shell Sort (kata)	0.326	11.92

Gambar 16. Pengujian data kata sebanyak 250000 data

```
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> gcc main.c sort.h -o sort
PS D:\USK\Semester 4\SDA\Tugas\tugas4\2308107010060_Tugas4_SDA_2025\src> ./sort
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 500000

Menjalankan pengurutan untuk 500000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	3709.867	23.84
Selection Sort (kata)	2639.003	23.84
Insertion Sort (kata)	1516.187	23.84
Merge Sort (kata)	0.248	23.84
Quick Sort (kata)	0.177	23.84
Shell Sort (kata)	0.679	23.84

Gambar 17. Pengujian data kata sebanyak 500000 data

```
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 1500000

Menjalankan pengurutan untuk 1500000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	6655.32	21.5
Selection Sort (kata)	4963.54	21.5
Insertion Sort (kata)	2539.90	21.5
Merge Sort (kata)	0.652	21.5
Quick Sort (kata)	0.414	21.5
Shell Sort (kata)	0.747	21.5

Gambar 18. Pengujian data kata sebanyak 1000000 data

```
=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 1000000

Menjalankan pengurutan untuk 1000000 data...

Hasil Pengurutan:
```

Algoritma yang Digunakan	Waktu yang Dibutuhkan (Detik)	Memori yang Digunakan (MB)
Bubble Sort (kata)	1985.064	4.21
Selection Sort (kata)	479.837	4.21
Insertion Sort (kata)	378.351	4.21
Merge Sort (kata)	0.216	4.21
Quick Sort (kata)	0.088	4.21
Shell Sort (kata)	0.212	4.21

Gambar 19. Pengujian data kata sebanyak 1500000 data

```

=== Pilih Tipe Data ===
1. Angka
2. Kata
Pilihan Anda [1-2]: 2
Masukkan jumlah data yang ingin diurutkan: 2000000

Menjalankan pengurutan untuk 2000000 data...

Hasil Pengurutan:

| Algoritma yang Digunakan | Waktu yang Dibutuhkan (Detik) | Memori yang Digunakan (MB) |
|-----|-----|-----|
| Bubble Sort (kata)      | 11831.638                      | 28.62                      |
| Selection Sort (kata)   | 8824.018                      | 28.62                      |
| Insertion Sort (kata)   | 4517.60                       | 28.62                      |
| Merge Sort (kata)       | 0.870                         | 28.62                      |
| Quick Sort (kata)       | 0.55                          | 28.62                      |
| Shell Sort (kata)       | 0.996                         | 28.62                      |

```

Gambar 20. Pengujian data kata sebanyak 2000000 data

TABEL HASIL EKSPERIMEN

1. Waktu

- Data Angka

Tabel 1. Hasil eksperimen data angka berdasarkan waktu

Data	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Shell Sort
10000	0.175	0.083	0.063	0.001	0.001	0.002
50000	7.302	1.802	1.227	0.008	0.005	0.012
100000	25.856	6.351	4.861	0.019	0.019	0.024
250000	154.343	40.895	30.631	0.043	0.028	0.057
500000	611.334	160.071	122.317	0.078	0.061	0.116
1000000	1887.601	475.842	377.855	0.215	0.089	0.212
1500000	4245.75	1069.69	848.25	0.425	0.220	0.565
2000000	7548.45	1897.103	1508.80	0.451	0.187	0.453

- Data Kata

Tabel 2. Hasil eksperimen data kata berdasarkan waktu

Data	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Shell Sort
10000	0.883	0.120	0.000	0.009	1.021	0.000
50000	10.843	4.564	2.548	0.016	0.008	0.023
100000	44.759	19.454	10.412	0.043	0.022	0.048
250000	739.448	551.504	281.114	0.145	0.094	0.326
500000	3709.867	2639.003	1516.187	0.248	0.177	0.679
1000000	1985.064	479.837	378.351	0.216	0.088	0.212
1500000	6655.32	4963.54	2539.90	0.652	0.414	0.747
2000000	11831.638	8824.018	4517.60	0.870	0.55	0.996

2. Memori

- Data Angka

Tabel 3. Hasil eksperimen data angka berdasarkan memori

Data	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Shell Sort
10000	0.04	0.04	0.04	0.04	0.04	0.04
50000	0.19	0.19	0.19	0.19	0.19	0.19
100000	0.38	0.38	0.38	0.38	0.38	0.38
250000	0.95	0.95	0.95	0.95	0.95	0.95
500000	1.91	1.91	1.91	1.91	1.91	1.91
1000000	3.81	3.81	3.81	3.81	3.81	3.81
1500000	12.62	12.62	12.62	12.62	12.62	12.62
2000000	16.1	16.1	16.1	16.1	16.1	16.1

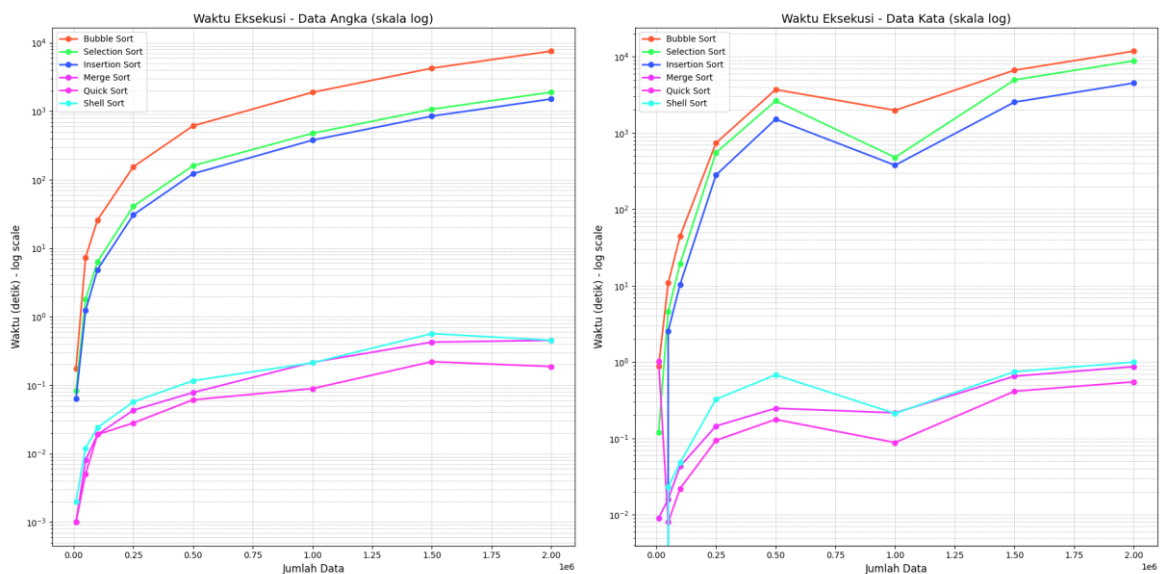
- Data Kata

Tabel 4. Hasil eksperimen data kata berdasarkan memori

Data	Bubble Sort	Selection Sort	Insertion Sort	Merge Sort	Quick Sort	Shell Sort
10000	0.48	0.48	0.48	0.48	0.48	0.48
50000	2.38	2.38	2.38	2.38	2.38	2.38
100000	4.77	4.77	4.77	4.77	4.77	4.77
250000	11.92	11.92	11.92	11.92	11.92	11.92
500000	23.84	23.84	23.84	23.84	23.84	23.84
1000000	4.21	4.21	4.21	4.21	4.21	4.21
1500000	21.5	21.5	21.5	21.5	21.5	21.5
2000000	28.62	28.62	28.62	28.62	28.62	28.62

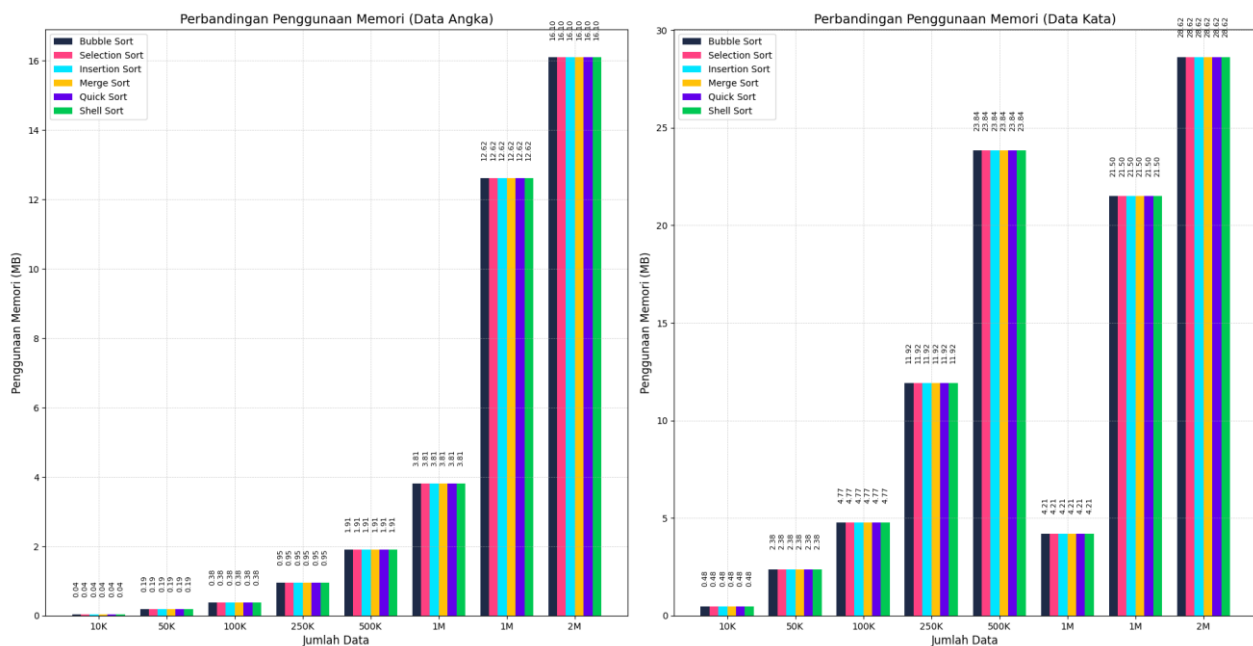
GRAFIK PERBANDINGAN

- Waktu



Gambar 21. Grafik Perbandingan berdasarkan Waktu antara data angka dan kata

- Memori



Gambar 22. Grafik Perbandingan berdasarkan memori antara data angka dan kata

ANALISIS PERFORMA ALGORITMA SORTING

Berdasarkan hasil pengamatan grafik waktu eksekusi, terlihat adanya perbedaan kinerja yang jelas di antara algoritma-algoritma pengurutan. Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu yang sangat tajam ketika jumlah data bertambah, terutama saat melampaui 100.000 record. Sementara itu, Merge Sort dan Quick Sort tetap menunjukkan waktu proses yang relatif rendah meskipun menghadapi data dalam jumlah besar.

Pengujian menunjukkan bahwa pengurutan kata membutuhkan waktu 2-10 kali lebih lama dibandingkan pengurutan angka, disebabkan oleh proses perbandingan string yang lebih rumit dan kebutuhan memori yang lebih besar.

Dalam grafik terlihat beberapa penurunan waktu eksekusi yang tidak terduga pada titik-titik tertentu. Kemungkinan besar ini terjadi karena fluktuasi penggunaan memori saat pengujian yang hanya difokuskan pada VSCode tanpa mengisolasi proses lainnya. Sistem operasi yang mengelola memori secara dinamis, aktivitas garbage collection, dan pembagian sumber daya dengan proses latar belakang turut memengaruhi ketidakkonsistenan hasil. Khusus untuk

data kata, terdapat lonjakan penggunaan memori yang tidak linear, yang mungkin disebabkan oleh kompresi memori atau alokasi memori tidak kontinu dari lingkungan runtime.

Pengurutan data kata membutuhkan waktu yang lebih lama dibandingkan data angka untuk semua algoritma. Bubble Sort memerlukan waktu hingga 11.831,64 detik (lebih dari 3 jam) untuk mengurutkan 2 juta data kata. Pola yang sama terlihat pada algoritma lain, dengan Selection Sort membutuhkan 8.824,02 detik dan Insertion Sort 4.517,60 detik. Algoritma $O(n \log n)$ tetap unggul dengan Quick Sort (0,55 detik), Merge Sort (0,870 detik), dan Shell Sort (0,996 detik).

KESIMPULAN

Hasil eksperimen ini membuktikan bahwa pemilihan algoritma pengurutan yang tepat sangat penting untuk efisiensi pemrosesan data. Algoritma dengan kompleksitas $O(n \log n)$ seperti Quick Sort, Merge Sort, dan Shell Sort secara konsisten menunjukkan performa superior dibandingkan algoritma $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort, terutama untuk data berukuran besar. Perbedaan kinerja ini semakin drastis seiring bertambahnya ukuran data, dengan algoritma $O(n^2)$ membutuhkan waktu hingga ribuan kali lebih lama untuk data berjumlah jutaan.

Jenis data yang diurutkan juga memberikan pengaruh signifikan terhadap performa algoritma. Pengurutan data kata membutuhkan waktu dan memori yang lebih besar dibandingkan data angka karena kompleksitas perbandingan string dan kebutuhan penyimpanan yang lebih tinggi. Untuk aplikasi praktis, Quick Sort merupakan pilihan optimal untuk kebanyakan kasus pengurutan karena konsistensi performanya yang tinggi pada berbagai ukuran dan jenis data. Namun, dalam situasi tertentu seperti data yang hampir terurut, algoritma seperti Insertion Sort atau Merge Sort bisa menjadi alternatif yang layak dipertimbangkan. Dapat disimpulkan, pentingnya analisis kompleksitas

algoritma dalam pengembangan perangkat lunak yang efisien, dan menunjukkan bahwa pemilihan algoritma yang tepat dapat menghemat waktu komputasi secara signifikan pada aplikasi yang melibatkan pengurutan data dalam jumlah besar.