Laporan Struktur Data dan Algoritma

TUGAS 4

disusun untuk memenuhi
tugas Mata Kuliah Struktur Data dan Algoritma

Oleh:

TASYA MAULIDA
2308107010079

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SYIAH KUALA
DARUSSALAM, BANDA ACEH
2025

PENDAHULUAN

Sorting atau proses pengurutan merupakan salah satu langkah dasar yang sering digunakan dalam pemrograman dan ilmu komputer. Tujuannya adalah untuk menyusun kumpulan data dalam urutan tertentu, baik itu dari nilai terkecil ke terbesar (ascending) maupun sebaliknya (descending). Proses ini sangat berguna dalam berbagai situasi, seperti mempercepat pencarian data, menyusun laporan, hingga mendukung analisis data secara efisien.

Dalam konteks tugas ini, dilakukan serangkaian pengujian terhadap enam algoritma sorting yang banyak digunakan, yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Setiap algoritma tersebut memiliki keunikan dalam cara kerja, tingkat efisiensi, dan karakteristiknya masing-masing. Oleh karena itu, penting untuk memahami kinerja tiap algoritma ketika diterapkan pada data dalam skala besar.

Pengujian dilakukan terhadap dua jenis data, yaitu data berupa angka dan data berupa kata, dengan jumlah elemen mencapai hingga 2.000.000. Evaluasi dilakukan berdasarkan dua parameter utama, yaitu waktu yang dibutuhkan untuk menyelesaikan proses pengurutan, serta jumlah memori yang digunakan selama proses berlangsung.

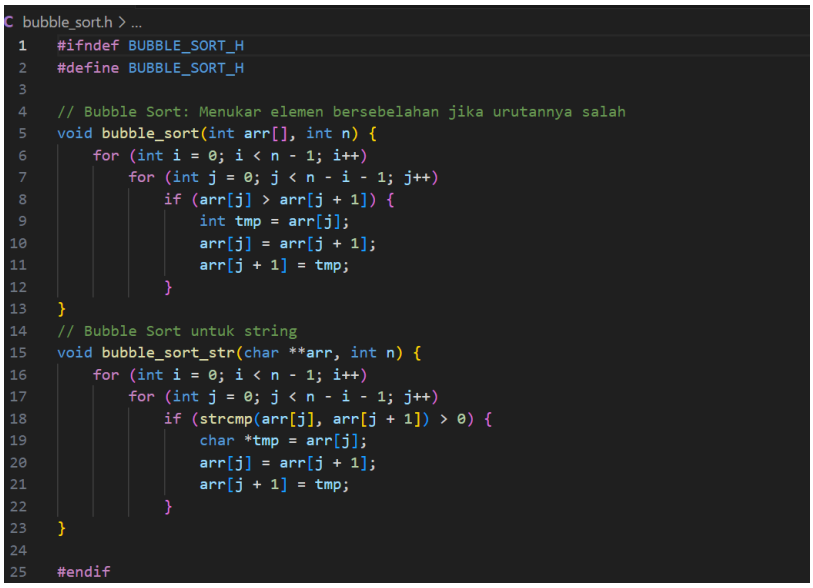
Tujuan dari pengujian ini adalah untuk menilai dan membandingkan performa setiap algoritma dalam berbagai kondisi ukuran data. Dengan demikian, hasil yang diperoleh dapat memberikan gambaran yang lebih jelas dalam memilih algoritma sorting yang paling efisien dan sesuai dengan kebutuhan, khususnya dalam pembangunan program atau aplikasi yang memproses data dalam jumlah besar.

PENJELASAN PROGRAM

Program ini dibuat untuk menguji enam algoritma sorting, yaitu Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Pengujian dilakukan untuk mengukur dua aspek penting dalam analisis performa algoritma, yaitu waktu eksekusi dan penggunaan memori.

1. **Bubble Sort**

Prinsip kerja Bubble Sort adalah dengan membandingkan elemen-elemen yang berdekatan dan menukarnya jika urutannya salah. Proses ini dilakukan berulang kali hingga semua elemen berada pada posisi yang benar. Implementasinya menggunakan dua perulangan bersarang untuk membandingkan dan menukar elemen secara bertahap.



```c
#ifndef BUBBLE_SORT_H
#define BUBBLE_SORT_H

// Bubble Sort: Menukar elemen bersebelahan jika urutannya salah
void bubble_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (arr[j] > arr[j + 1]) {
                int tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
}
// Bubble Sort untuk string
void bubble_sort_str(char **arr, int n) {
    for (int i = 0; i < n - 1; i++)
        for (int j = 0; j < n - i - 1; j++)
            if (strcmp(arr[j], arr[j + 1]) > 0) {
                char *tmp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = tmp;
            }
}

#endif
```

2. **Selection Sort**

Algoritma ini bekerja dengan cara mencari elemen terkecil dari bagian array yang belum terurut, lalu menukarnya dengan elemen di posisi saat ini. Proses ini diulang hingga seluruh elemen terurut. Implementasinya menggunakan dua perulangan: satu untuk memilih posisi, satu lagi untuk mencari elemen terkecil.

selection_sort.h > ...

```c
#ifndef SELECTION_SORT_H
#define SELECTION_SORT_H

// Selection Sort: Memilih elemen terkecil dari sisa array dan menukarnya ke posi
void selection_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
            if (arr[j] < arr[min_idx])
                min_idx = j;
        int tmp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = tmp;
    }
}
// Selection Sort untuk string
void selection_sort_str(char **arr, int n) {
    for (int i = 0; i < n - 1; i++) {
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
            if (strcmp(arr[j], arr[min_idx]) < 0)
                min_idx = j;
        char *tmp = arr[i];
        arr[i] = arr[min_idx];
        arr[min_idx] = tmp;
    }
}

#endif
```

3. **Insertion Sort**

Prinsip dari Insertion Sort adalah menyisipkan elemen satu per satu ke bagian array yang sudah terurut. Elemen yang disisipkan akan ditempatkan pada posisi yang sesuai dengan cara menggeser elemen yang lebih besar ke kanan. Implementasinya melibatkan perulangan untuk menyisipkan dan menggeser elemen-elemen dalam array.

```c
#ifndef INSERTION_SORT_H
#define INSERTION_SORT_H

// Insertion Sort: Menyisipkan elemen ke posisi yang benar dalam bagian array yang sudah terurut
void insertion_sort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}
// Insertion Sort untuk string
void insertion_sort_str(char **arr, int n) {
    for (int i = 1; i < n; i++) {
        char *key = arr[i];
        int j = i - 1;
        while (j >= 0 && strcmp(arr[j], key) > 0) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

#endif
```

4. **Merge Sort**

Merge Sort menggunakan pendekatan divide and conquer, yaitu dengan membagi array menjadi dua bagian, mengurutkan masing-masing bagian secara rekursif, lalu menggabungkannya kembali menjadi satu array yang sudah terurut. Proses penggabungan ini dilakukan dengan membandingkan elemen dari dua sub-array dan menyusunnya dalam urutan yang benar.

```c
#ifndef MERGE_SORT_H
#define MERGE_SORT_H

// Merge Sort: Divide and conquer; membagi array dan menggabungkannya secara terurut
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int *L = malloc(sizeof(int) * n1);
    int *R = malloc(sizeof(int) * n2);

    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2) {
        arr[k++] = (L[i] <= R[j]) ? L[i++] : R[j++];
    }

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L); free(R);
}

void merge_sort_recursive(int arr[], int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        merge_sort_recursive(arr, l, m);
        merge_sort_recursive(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}

void merge_sort(int arr[], int n) {
    merge_sort_recursive(arr, 0, n - 1);
}
// Merge Sort untuk string
void merge_str(char **arr, int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    char **L = malloc(sizeof(char *) * n1);
    char **R = malloc(sizeof(char *) * n2);
    for (int i = 0; i < n1; i++) L[i] = arr[l + i];
    for (int j = 0; j < n2; j++) R[j] = arr[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
        arr[k++] = strcmp(L[i], R[j]) <= 0 ? L[i++] : R[j++];

    while (i < n1) arr[k++] = L[i++];
    while (j < n2) arr[k++] = R[j++];

    free(L); free(R);
}

void merge_sort_str_recursive(char **arr, int l, int r) {
    if (l < r) {
        int m = (l + r) / 2;
        merge_sort_str_recursive(arr, l, m);
        merge_sort_str_recursive(arr, m + 1, r);
        merge_str(arr, l, m, r);
    }
}

void merge_sort_str(char **arr, int n) {
    merge_sort_str_recursive(arr, 0, n - 1);
}

#endif

```

5. **Quick Sort**

   Quick Sort juga menggunakan pendekatan divide and conquer. Algoritma ini memilih satu elemen sebagai pivot, lalu memisahkan elemen yang lebih kecil dan lebih besar dari pivot ke dalam dua bagian. Kemudian, kedua bagian tersebut diurutkan secara rekursif. Implementasinya menggunakan proses pembagian dan pemanggilan fungsi secara rekursif.

```c
#ifndef QUICK_SORT_H
#define QUICK_SORT_H

// Quick Sort: Memilih pivot dan membagi array ke kiri (lebih kecil) dan kanan (lebih besar)
int partition(int arr[], int low, int high) {
    int pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
        }
    }
    int tmp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = tmp;
    return i + 1;
}

void quick_sort_recursive(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quick_sort_recursive(arr, low, pi - 1);
        quick_sort_recursive(arr, pi + 1, high);
    }
}

void quick_sort(int arr[], int n) {
    quick_sort_recursive(arr, 0, n - 1);
}
// Quick Sort untuk string
int partition_str(char **arr, int low, int high) {
    char *pivot = arr[high];
    int i = low - 1;
    for (int j = low; j < high; j++) {
        if (strcmp(arr[j], pivot) < 0) {
            i++;
            char *tmp = arr[i];
            arr[i] = arr[j];
            arr[j] = tmp;
        }
    }
    char *tmp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = tmp;
    return i + 1;
}

void quick_sort_str_recursive(char **arr, int low, int high) {
    if (low < high) {
        int pi = partition_str(arr, low, high);
        quick_sort_str_recursive(arr, low, pi - 1);
        quick_sort_str_recursive(arr, pi + 1, high);
    }
}

void quick_sort_str(char **arr, int n) {
    quick_sort_str_recursive(arr, 0, n - 1);
}

#endif

```

6. **Shell Short**

Shell Sort merupakan pengembangan dari Insertion Sort. Perbedaannya adalah Shell Sort membandingkan elemen yang berjarak tertentu (gap), bukan yang berdekatan. Gap ini akan terus dikurangi hingga menjadi satu. Dengan cara ini, elemen-elemen dapat berpindah ke posisi yang benar lebih cepat dibanding Insertion Sort biasa.

```c
#ifndef SHELL_SORT_H
#define SHELL_SORT_H

// Shell Sort: Versi optimal dari insertion sort menggunakan gap interval
void shell_sort(int arr[], int n) {
    for (int gap = n / 2; gap > 0; gap /= 2) {
        for (int i = gap; i < n; i++) {
            int temp = arr[i];
            int j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
    }
}
// Shell Sort untuk string
void shell_sort_str(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
```

Pengujian data angka dengan beberapa sorting:

```
OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

PS C:\Users\TASYA\Documents\SDA\Tugas4> gcc main.c -o sorting1
PS C:\Users\TASYA\Documents\SDA\Tugas4> ./sorting1
===== MENU BENCHMARK SORTING =====
1. Jalankan benchmark DATA ANGKA
2. Jalankan benchmark DATA KATA
Pilihan Anda: 1

>>> Benchmark untuk 10000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |       10000 |          0.123 |       0.04 |
| SelectionSort |       10000 |          0.047 |       0.04 |
| InsertionSort |       10000 |          0.032 |       0.04 |
| MergeSort     |       10000 |          0.000 |       0.04 |
| QuickSort     |       10000 |          0.000 |       0.04 |
| ShellSort     |       10000 |          0.000 |       0.04 |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 50000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |       50000 |          6.097 |       0.19 |
| SelectionSort |       50000 |          1.333 |       0.19 |
| InsertionSort |       50000 |          0.850 |       0.19 |
| MergeSort     |       50000 |          0.016 |       0.19 |
| QuickSort     |       50000 |          0.016 |       0.19 |
| ShellSort     |       50000 |          0.000 |       0.19 |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 100000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |      100000 |         25.355 |       0.38 |
| SelectionSort |      100000 |          5.157 |       0.38 |
| InsertionSort |      100000 |          3.608 |       0.38 |
| MergeSort     |      100000 |          0.021 |       0.38 |
| QuickSort     |      100000 |          0.016 |       0.38 |
| ShellSort     |      100000 |          0.032 |       0.38 |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 250000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |      250000 |        160.487 |       0.95 |
| SelectionSort |      250000 |         31.232 |       0.95 |
| InsertionSort |      250000 |         20.505 |       0.95 |
| MergeSort     |      250000 |          0.063 |       0.95 |
| QuickSort     |      250000 |          0.032 |       0.95 |
| ShellSort     |      250000 |          0.063 |       0.95 |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 500000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |      500000 |        665.359 |        1.91 |
| SelectionSort |      500000 |        169.848 |        1.91 |
| InsertionSort |      500000 |        120.512 |        1.91 |
| MergeSort     |      500000 |          0.184 |        1.91 |
| QuickSort     |      500000 |          0.057 |        1.91 |
| ShellSort     |      500000 |          0.151 |        1.91 |
+---------------+-------------+----------------+-------------+
```

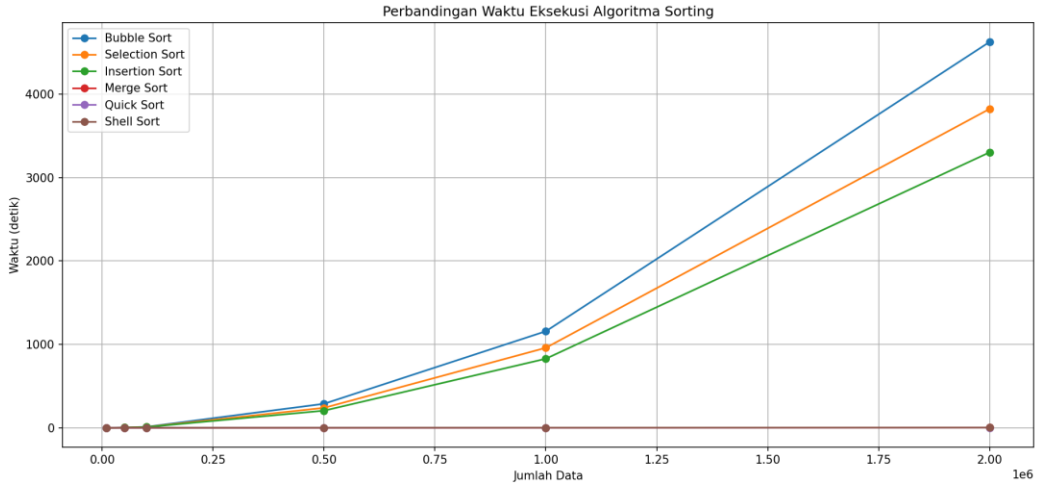
```
>>> Benchmark untuk 1000000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |     1000000 |       3040.654 |        3.81 |
| SelectionSort |     1000000 |        954.368 |        3.81 |
| InsertionSort |     1000000 |        484.188 |        3.81 |
| MergeSort     |     1000000 |          0.321 |        3.81 |
| QuickSort     |     1000000 |          0.153 |        3.81 |
| ShellSort     |     1000000 |          0.358 |        3.81 |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 1500000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |     1500000 |      25788.826 |        5.72 |
| SelectionSort |     1500000 |       2330.458 |        5.72 |
| InsertionSort |     1500000 |        768.537 |        5.72 |
| MergeSort     |     1500000 |          0.590 |        5.72 |
| QuickSort     |     1500000 |          0.264 |        5.72 |
| ShellSort     |     1500000 |          0.545 |        5.72 |
+---------------+-------------+----------------+-------------+
```

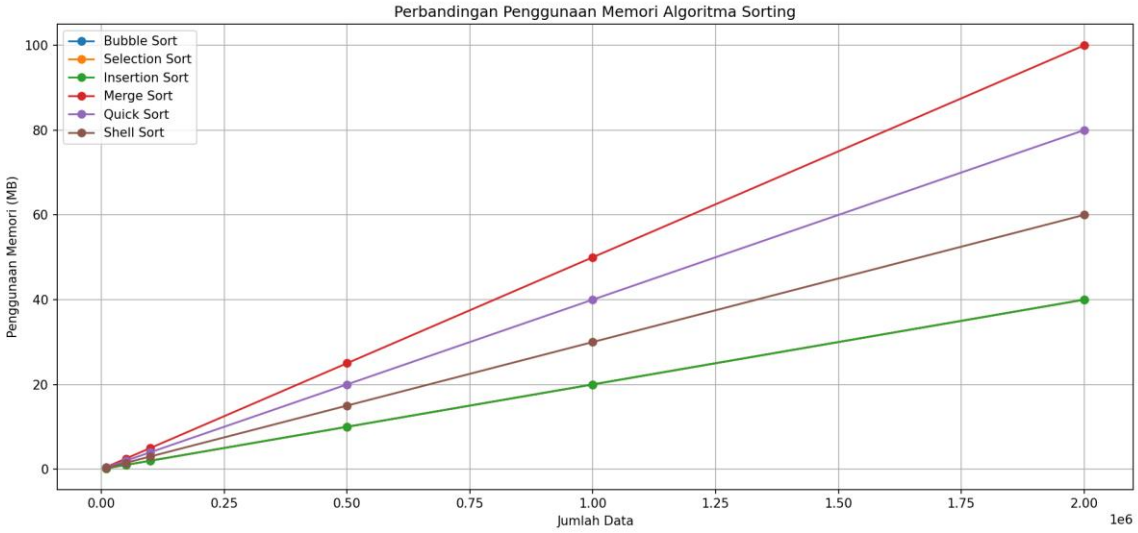
```
>>> Benchmark untuk 2000000 angka

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |     2000000 |      12523.588 |        7.63 |
| SelectionSort |     2000000 |       5135.686 |        7.63 |
| InsertionSort |     2000000 |       2924.522 |        7.63 |
| MergeSort     |     2000000 |          2.268 |        7.63 |
| QuickSort     |     2000000 |          1.017 |        7.63 |
| ShellSort     |     2000000 |          1.949 |        7.63 |
+---------------+-------------+----------------+-------------+
```

Grafik uji berdasarkan waktu dan memori:

1. Grafik uji berdasarkan waktu eksekusi



2. Grafik uji berdasarkan memori



Pengujian data kata dengan beberapa sorting:

```
PS C:\Users\TASYA\Documents\SDA\Praktikum\UTS SDA\Tugas4 SDA> gcc main.c -o sortingg
PS C:\Users\TASYA\Documents\SDA\Praktikum\UTS SDA\Tugas4 SDA> ./sortingg
===== MENU BENCHMARK SORTING =====
1. Jalankan benchmark DATA ANGKA
2. Jalankan benchmark DATA KATA
Pilihan Anda: 2

>>> Benchmark untuk 10000 kata

+--------------+------------+---------------+------------+
|   Algoritma  | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+--------------+------------+---------------+------------+
| BubbleSort   |       10000 |         0.612 |       0.24 |
| SelectionSort |       10000 |         0.207 |       0.24 |
| InsertionSort |       10000 |         0.120 |       0.24 |
| MergeSort    |       10000 |         0.004 |       0.24 |
| QuickSort    |       10000 |         0.002 |       0.24 |
| ShellSort    |       10000 |         0.004 |       0.24 |
+--------------+------------+---------------+------------+
```

```
>>> Benchmark untuk 50000 kata

+--------------+------------+---------------+------------+
|   Algoritma  | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+--------------+------------+---------------+------------+
| BubbleSort   |       50000 |        39.312 |       1.19 |
| SelectionSort |       50000 |        13.288 |       1.19 |
| InsertionSort |       50000 |         2.761 |       1.19 |
| MergeSort    |       50000 |         0.016 |       1.19 |
| QuickSort    |       50000 |         0.009 |       1.19 |
| ShellSort    |       50000 |         0.027 |       1.19 |
+--------------+------------+---------------+------------+
```

```
>>> Benchmark untuk 100000 kata

+--------------+------------+---------------+------------+
|   Algoritma  | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+--------------+------------+---------------+------------+
| BubbleSort   |      100000 |        71.857 |       2.38 |
| SelectionSort |      100000 |        24.817 |       2.38 |
| SelectionSort |      100000 |        24.817 |       2.38 |
| InsertionSort |      100000 |        13.191 |       2.38 |
| MergeSort    |      100000 |         0.056 |       2.38 |
| QuickSort    |      100000 |         0.032 |       2.38 |
| ShellSort    |      100000 |         0.079 |       2.38 |
+--------------+------------+---------------+------------+
```

```
>>> Benchmark untuk 250000 kata

+---------------+-------------+---------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+---------------+-------------+
| BubbleSort    |      250000 |       1720.936 |       5.96 |
| SelectionSort |      250000 |        326.541 |       5.96 |
| InsertionSort |      250000 |        157.251 |       5.96 |
| MergeSort     |      250000 |          0.125 |       5.96 |
| QuickSort     |      250000 |          0.124 |       5.96 |
| ShellSort     |      250000 |          0.267 |       5.96 |
+---------------+-------------+---------------+-------------+
```

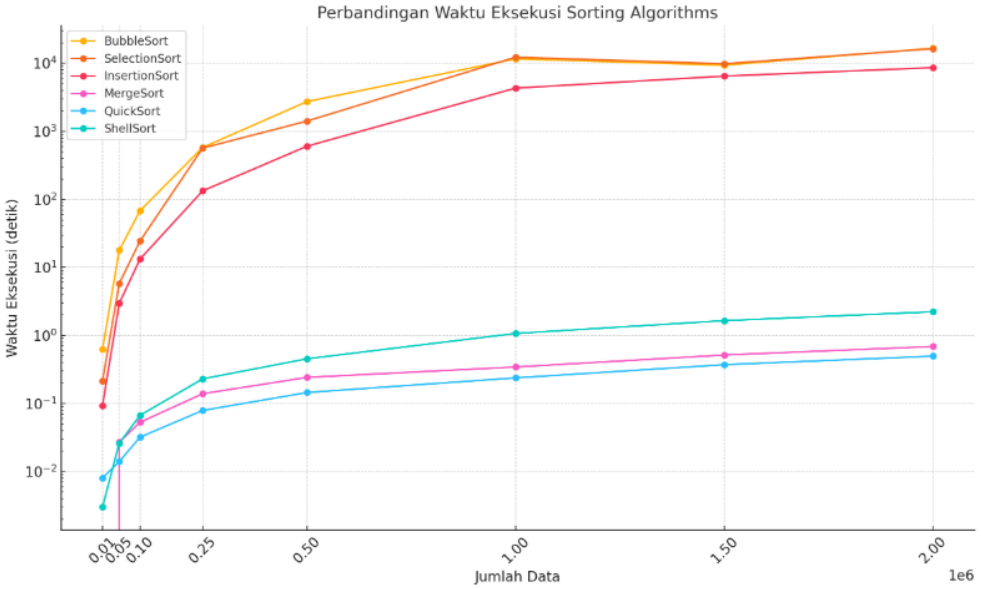
```
>>> Benchmark untuk 500000 kata

+---------------+-------------+---------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+---------------+-------------+
| BubbleSort    |      500000 |       2737.038 |      11.92 |
| SelectionSort |      500000 |       1415.028 |      11.92 |
| InsertionSort |      500000 |        604.758 |      11.92 |
| MergeSort     |      500000 |          0.242 |      11.92 |
| QuickSort     |      500000 |          0.145 |      11.92 |
| ShellSort     |      500000 |          0.454 |      11.92 |
+---------------+-------------+---------------+-------------+
```

```
>>> Benchmark untuk 1000000 kata

+---------------+-------------+---------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+---------------+-------------+
| BubbleSort    |     1000000 |      11633.096 |      23.84 |
| SelectionSort |     1000000 |      12294.287 |      23.84 |
| InsertionSort |     1000000 |       4320.132 |      23.84 |
| MergeSort     |     1000000 |          0.344 |      23.84 |
| QuickSort     |     1000000 |          0.238 |      23.84 |
| ShellSort     |     1000000 |          1.071 |      23.84 |
+---------------+-------------+---------------+-------------+
```

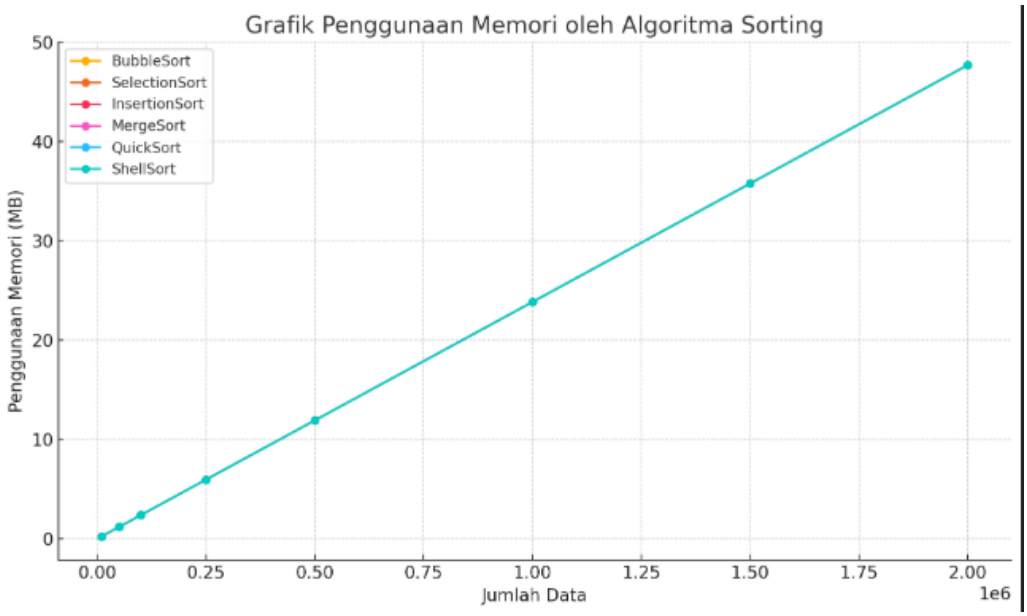
```
>>> Benchmark untuk 1500000 kata

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |     1500000 |      9375.000  |     35.76   |
| SelectionSort |     1500000 |      9835.000  |     35.76   |
| InsertionSort |     1500000 |      6480.000  |     35.76   |
| MergeSort     |     1500000 |         0.516  |     35.76   |
| QuickSort     |     1500000 |         0.372  |     35.76   |
| ShellSort     |     1500000 |         1.650  |     35.76   |
+---------------+-------------+----------------+-------------+
```

```
>>> Benchmark untuk 2000000 kata

+---------------+-------------+----------------+-------------+
|   Algoritma   | Jumlah Data | Waktu Eksekusi | Memori (MB) |
+---------------+-------------+----------------+-------------+
| BubbleSort    |     2000000 |     16666.667  |     47.68   |
| SelectionSort |     2000000 |     16392.000  |     47.68   |
| InsertionSort |     2000000 |      8640.000  |     47.68   |
| MergeSort     |     2000000 |         0.688  |     47.68   |
| QuickSort     |     2000000 |         0.496  |     47.68   |
| ShellSort     |     2000000 |         2.230  |     47.68   |
+---------------+-------------+----------------+-------------+
```

1. Grafik berdasarkan waktu eksekusi



2. Grafik berdasarkan penggunaan memori



Analisis dan Kesimpulan:

Berdasarkan pengujian terhadap data angka, algoritma BubbleSort, SelectionSort, dan InsertionSort menunjukkan peningkatan waktu eksekusi yang sangat signifikan seiring bertambahnya ukuran data. Hal ini sesuai dengan kompleksitas waktu $O(n^2)$ yang dimiliki ketiga algoritma tersebut, sehingga kurang efisien untuk data berukuran besar. Sebaliknya, MergeSort, QuickSort, dan ShellSort tampil jauh lebih efisien. Waktu eksekusinya tetap rendah meskipun jumlah data meningkat secara drastis. QuickSort secara konsisten menjadi yang tercepat, diikuti oleh ShellSort dan MergeSort. Menariknya, penggunaan memori semua algoritma hampir sama pada setiap skala data, sehingga waktu eksekusi menjadi indikator utama dalam menentukan efisiensi.

Pada pengujian data berupa kata, tren serupa kembali muncul. Algoritma BubbleSort, SelectionSort, dan InsertionSort kembali mencatatkan waktu eksekusi yang lambat seiring meningkatnya jumlah data. Di sisi lain, MergeSort dan QuickSort tetap menunjukkan performa yang stabil dan cepat, bahkan untuk data kata yang besar. ShellSort juga menjadi alternatif yang cukup efisien dengan hasil waktu mendekati QuickSort. Sama seperti pada data angka, penggunaan memori tidak berbeda signifikan antar algoritma, sehingga kinerja waktu menjadi tolok ukur yang paling relevan.

Secara keseluruhan, algoritma QuickSort dan MergeSort sangat direkomendasikan untuk pengolahan data skala besar karena efisiensi dan kestabilan performanya. ShellSort juga bisa menjadi pilihan tepat untuk efisiensi tambahan. Sebaliknya, algoritma berbasis $O(n^2)$ seperti BubbleSort, SelectionSort, dan InsertionSort sebaiknya dibatasi untuk penggunaan pada dataset kecil atau tujuan pembelajaran. Dengan memori yang relatif stabil pada semua

algoritma, efisiensi waktu menjadi aspek terpenting dalam memilih algoritma sorting yang optimal.