Laporan Mata Kuliah Struktur Data dan Algoritma

TUGAS 4
IMPLEMENTASI DAN ANALISIS PERFORMA SORTING ALGORITM

disusun untuk memenuhi
tugas mata kuliah Struktur Data dan Algoritma

Oleh:

Nurul Izzati
2308108010047

JURUSAN INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

Dalam eksperimen ini, mahasiswa diminta untuk mengimplementasikan dan menganalisis performa dari enam algoritma sorting, yaitu: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort.

Uji coba dilakukan terhadap dua jenis data — angka dan kata, dengan jumlah data yang bervariasi hingga skala besar. Tujuan dari eksperimen ini adalah untuk memahami perbedaan kompleksitas waktu dan efisiensi memori dari masing-masing algoritma, serta mengidentifikasi algoritma yang paling optimal untuk kebutuhan pengurutan data dalam konteks nyata.

A. Deskripsi algoritma dan cara implementasinya

Setiap algoritma yang diuji memiliki karakteristik dan metode kerja yang berbeda. Berikut penjelasan singkat mengenai prinsip kerja dan cara implementasi dari masing-masing algoritma yang digunakan dalam eksperimen ini:

1. Bubble Sort

Bubble Sort adalah algoritma sederhana yang membandingkan elemen berdekatan dan menukarnya jika urutannya salah. Proses ini diulang hingga tidak ada lagi elemen yang perlu ditukar.

Cara Kerja:

- Bandingkan elemen ke-1 dan ke-2, tukar jika perlu.
- Lanjutkan ke elemen ke-2 dan ke-3, dst.
- Setelah satu putaran, elemen terbesar sudah berada di posisi akhir.
- Ulangi proses untuk sisa array.

Implementasi:

- Data angka

```c
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;
            }
}
```

- Data kata

```c
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;
            }
}
```

2. Selection Sort

Selection Sort mencari elemen terkecil dari array dan menukarnya dengan elemen di awal. Proses diulang untuk setiap posisi berikutnya hingga array terurut.

Cara Kerja:

- Temukan elemen minimum dari sisa array.
- Tukar dengan elemen pada posisi sekarang.
- Ulangi untuk seluruh array.

Implementasi:

- Data angka

```c
void selection_sort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++)
            if (arr[j] < arr[minIdx]) minIdx = j;
        int tmp = arr[i];
        arr[i] = arr[minIdx];
        arr[minIdx] = tmp;
    }
}
```

- Data kata

```c
void selection_sort_str(char *arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int minIdx = i;
        for (int j = i + 1; j < n; j++)
            if (strcmp(arr[j], arr[minIdx]) < 0) minIdx = j;
        char *tmp = arr[i];
        arr[i] = arr[minIdx];
        arr[minIdx] = tmp;
    }
}
```

3. Insertion Sort

Insertion Sort menyusun elemen satu per satu seperti menyusun kartu. Elemen baru dimasukkan ke posisi yang sesuai dalam array yang telah terurut.

Cara Kerja:

- Mulai dari indeks ke-1.
- Bandingkan elemen dengan elemen sebelum-sebelumnya.
- Geser elemen lebih besar ke kanan.
- Masukkan elemen ke tempat yang sesuai.

Implementasi:

- Data angka

```c
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;
    }
}
```

- Data kata

```c
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;
    }
}
```

4. Merge Sort

Merge Sort menggunakan teknik divide and conquer: membagi array menjadi dua bagian, mengurutkan keduanya secara rekursif, lalu menggabungkan hasilnya.

Cara Kerja:

- Bagi array menjadi dua bagian.
- Urutkan kedua bagian secara rekursif.

- Gabungkan kedua array yang sudah terurut.

Implementasi:

- Data angka

```c
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    int *L = malloc(n1 * sizeof(int));
    int *R = malloc(n2 * sizeof(int));
    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(int arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        merge_sort(arr, l, m);
        merge_sort(arr, m + 1, r);
        merge(arr, l, m, r);
    }
}
```

- Data kata

```c
void merge_str(char *arr[], int l, int m, int r) {
    int n1 = m - l + 1, n2 = r - m;
    char **L = malloc(n1 * sizeof(char *));
    char **R = malloc(n2 * sizeof(char *));
    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(char *arr[], int l, int r) {
    if (l < r) {
        int m = l + (r - l) / 2;
        merge_sort_str(arr, l, m);
        merge_sort_str(arr, m + 1, r);
        merge_str(arr, l, m, r);
    }
}
```

5. Quick Sort

Quick Sort memilih satu elemen sebagai pivot dan membagi array ke dua bagian: yang lebih kecil dan yang lebih besar dari pivot. Lalu melakukan sort secara rekursif.

Cara Kerja:

- Pilih pivot (misalnya elemen terakhir).
- Partisi array ke kiri (lebih kecil) dan kanan (lebih besar).
- Sort kedua bagian secara rekursif.

Implementasi:

- Data angka

```c
int partition(int arr[], int low, int high) {
    int pivot = arr[high], i = low - 1;
    for (int j = low; j < high; j++)
        if (arr[j] <= pivot) {
            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(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quick_sort(arr, low, pi - 1);
        quick_sort(arr, pi + 1, high);
    }
}
```

- Data kata

```c
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) {
            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(char *arr[], int low, int high) {
    if (low < high) {
        int pi = partition_str(arr, low, high);
        quick_sort_str(arr, low, pi - 1);
        quick_sort_str(arr, pi + 1, high);
    }
}
```

6. Shell Sort

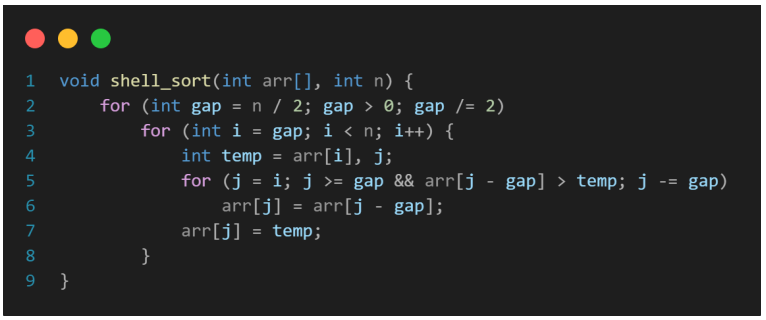
Shell Sort adalah versi peningkatan dari Insertion Sort dengan membandingkan elemen yang lebih jauh jaraknya terlebih dahulu. Jarak antar elemen (gap) dikurangi secara bertahap.

Cara Kerja:

- Pilih gap awal (biasanya n/2), lalu kurangi perlahan.
- Lakukan insertion sort untuk elemen-elemen dengan jarak gap.
- Ulangi hingga gap = 1.

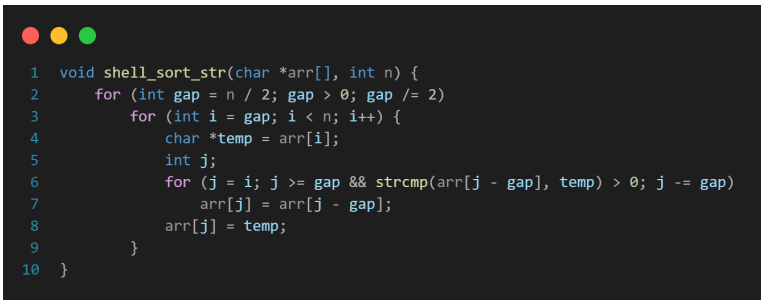
Implementasi:

- Data angka



```c
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], j;
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
                arr[j] = arr[j - gap];
            arr[j] = temp;
        }
}
```

- Data kata

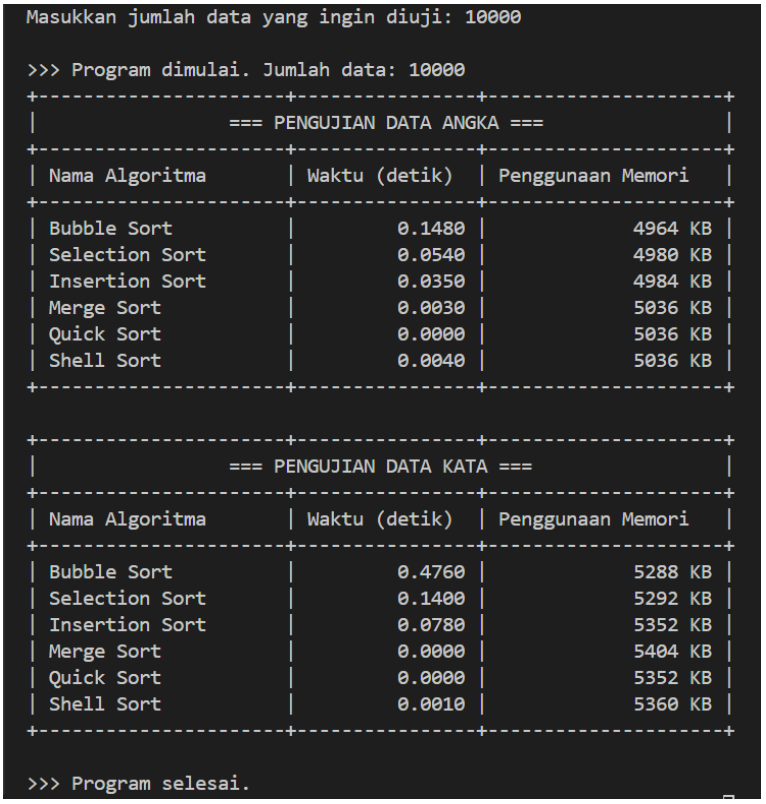


```c
void shell_sort_str(char *arr[], int n) {
    for (int gap = n / 2; gap > 0; gap /= 2)
        for (int i = gap; i < n; i++) {
            char *temp = arr[i];
            int j;
            for (j = i; j >= gap && strcmp(arr[j - gap], temp) > 0; j -= gap)
                arr[j] = arr[j - gap];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

B. Tabel hasil eksperimen (waktu dan memori)

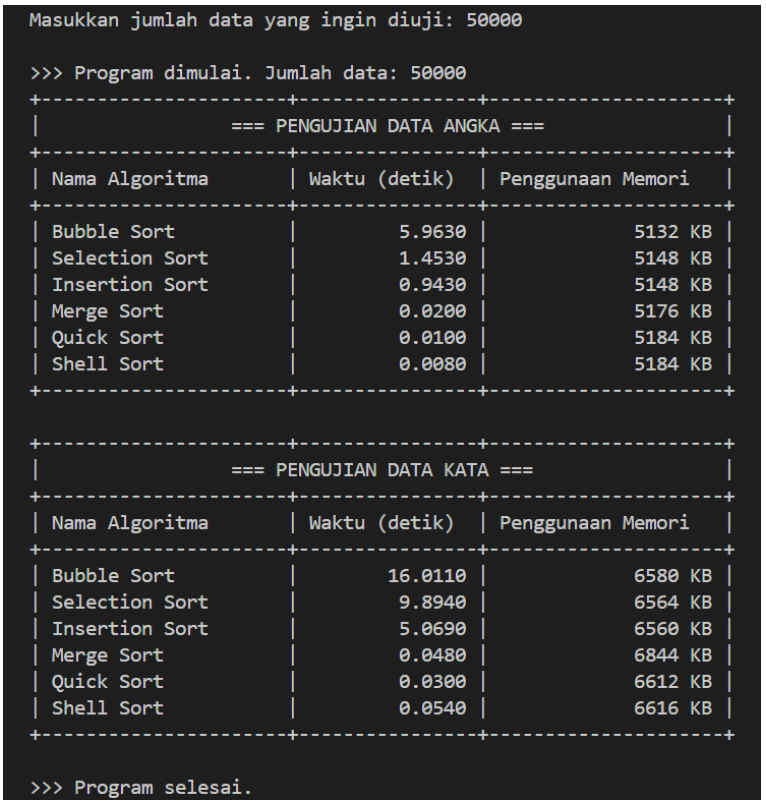
Berikut adalah tabel hasil eksperimen pengujian beberapa algoritma dengan jumlah data yang berbeda-beda yaitu untuk data dengan jumlah 10.000, 50.000, 100.000, 250.000, 500.000, 1.000.000, 1.500.000 dan 2.000.000. Data yang digunakan untuk pengujian ini didapatkan dengan generate angka dan data sebanyak 20.000.000 data untuk masing-masing data yang disimpan dalam data_angka.txt dan juga data_kata.txt.

- Pengujian dengan jumlah data 10.000



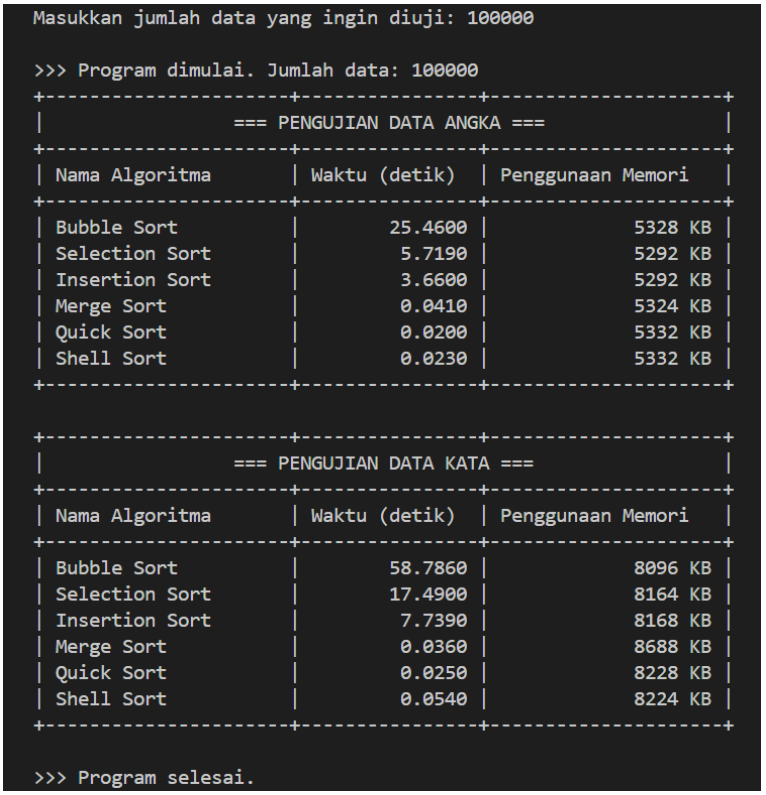
```
Masukkan jumlah data yang ingin diuji: 10000

>>> Program dimulai. Jumlah data: 10000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |        0.1480 |            4964 KB |
| Selection Sort       |        0.0540 |            4980 KB |
| Insertion Sort       |        0.0350 |            4984 KB |
| Merge Sort           |        0.0030 |            5036 KB |
| Quick Sort           |        0.0000 |            5036 KB |
| Shell Sort           |        0.0040 |            5036 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |        0.4760 |            5288 KB |
| Selection Sort       |        0.1400 |            5292 KB |
| Insertion Sort       |        0.0780 |            5352 KB |
| Merge Sort           |        0.0000 |            5404 KB |
| Quick Sort           |        0.0000 |            5352 KB |
| Shell Sort           |        0.0010 |            5360 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 50.000



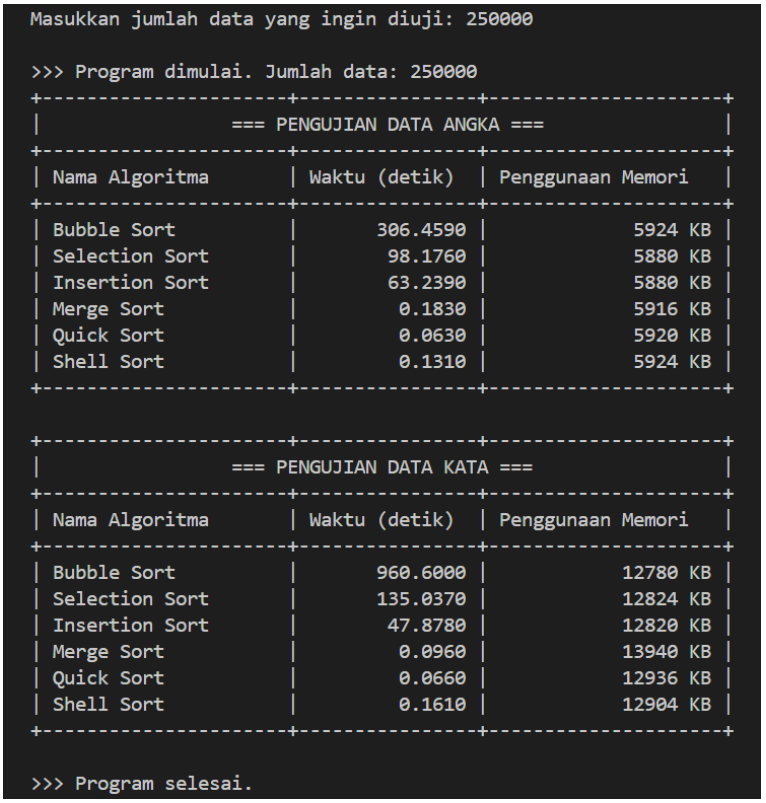
```
Masukkan jumlah data yang ingin diuji: 50000

>>> Program dimulai. Jumlah data: 50000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |        5.9630 |            5132 KB |
| Selection Sort       |        1.4530 |            5148 KB |
| Insertion Sort       |        0.9430 |            5148 KB |
| Merge Sort           |        0.0200 |            5176 KB |
| Quick Sort           |        0.0100 |            5184 KB |
| Shell Sort           |        0.0080 |            5184 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |       16.0110 |            6580 KB |
| Selection Sort       |        9.8940 |            6564 KB |
| Insertion Sort       |        5.0690 |            6560 KB |
| Merge Sort           |        0.0480 |            6844 KB |
| Quick Sort           |        0.0300 |            6612 KB |
| Shell Sort           |        0.0540 |            6616 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 100.000



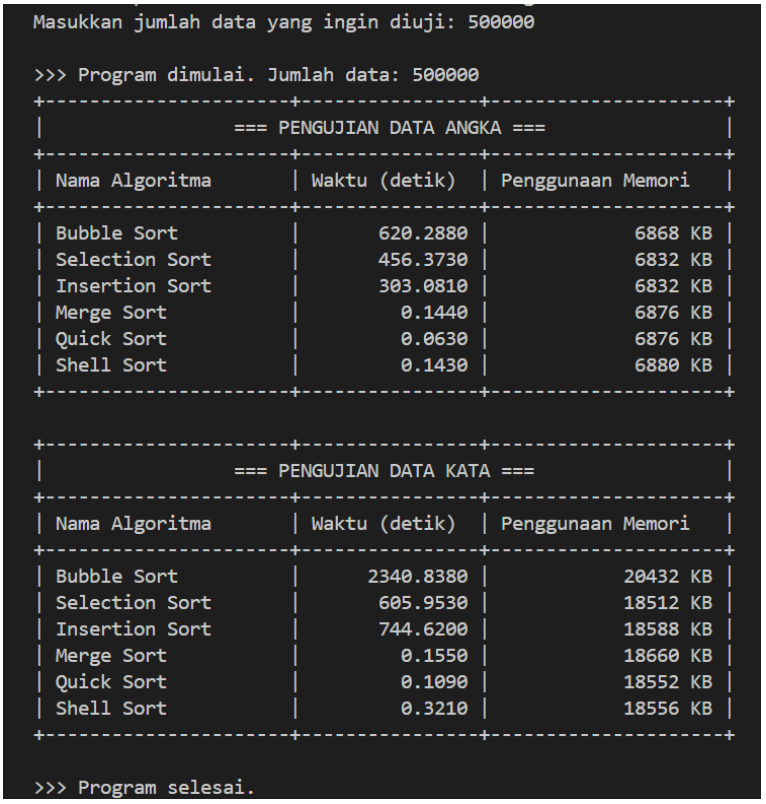
```
Masukkan jumlah data yang ingin diuji: 100000

>>> Program dimulai. Jumlah data: 100000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |       25.4600 |            5328 KB |
| Selection Sort       |        5.7190 |            5292 KB |
| Insertion Sort       |        3.6600 |            5292 KB |
| Merge Sort           |        0.0410 |            5324 KB |
| Quick Sort           |        0.0200 |            5332 KB |
| Shell Sort           |        0.0230 |            5332 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |       58.7860 |            8096 KB |
| Selection Sort       |       17.4900 |            8164 KB |
| Insertion Sort       |        7.7390 |            8168 KB |
| Merge Sort           |        0.0360 |            8688 KB |
| Quick Sort           |        0.0250 |            8228 KB |
| Shell Sort           |        0.0540 |            8224 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 250.000



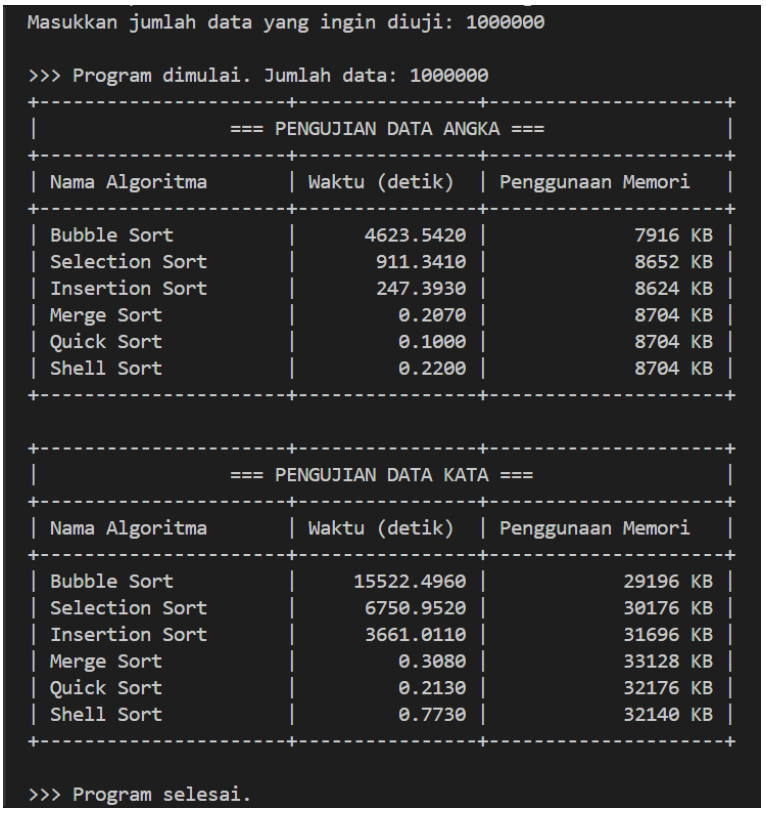
```
Masukkan jumlah data yang ingin diuji: 250000

>>> Program dimulai. Jumlah data: 250000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |      306.4590 |            5924 KB |
| Selection Sort       |       98.1760 |            5880 KB |
| Insertion Sort       |       63.2390 |            5880 KB |
| Merge Sort           |        0.1830 |            5916 KB |
| Quick Sort           |        0.0630 |            5920 KB |
| Shell Sort           |        0.1310 |            5924 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |      960.6000 |           12780 KB |
| Selection Sort       |      135.0370 |           12824 KB |
| Insertion Sort       |       47.8780 |           12820 KB |
| Merge Sort           |        0.0960 |           13940 KB |
| Quick Sort           |        0.0660 |           12936 KB |
| Shell Sort           |        0.1610 |           12904 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 500.000



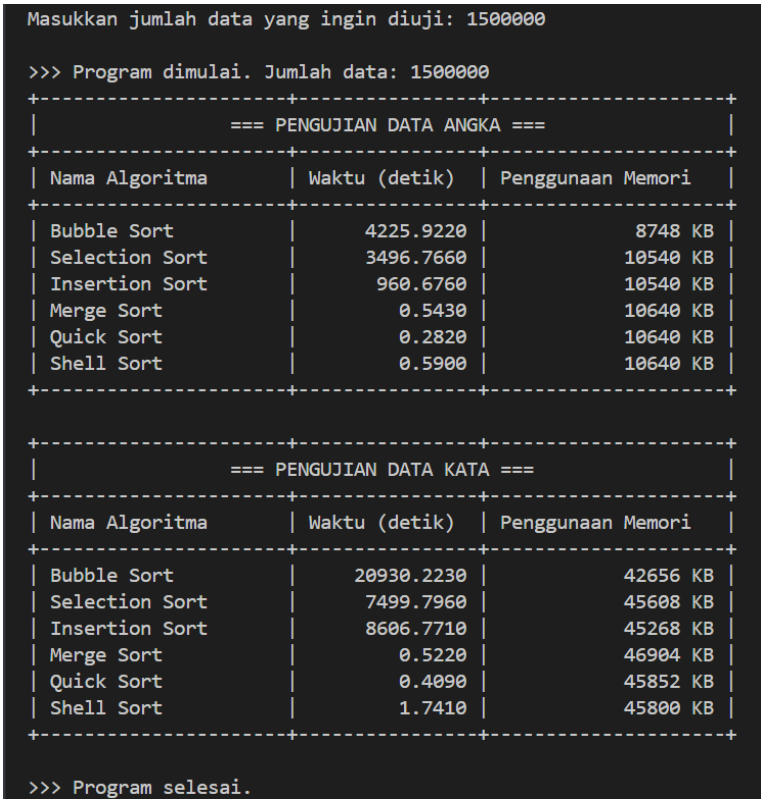
```
Masukkan jumlah data yang ingin diuji: 500000

>>> Program dimulai. Jumlah data: 500000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===               |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |      620.2880 |            6868 KB |
| Selection Sort       |      456.3730 |            6832 KB |
| Insertion Sort       |      303.0810 |            6832 KB |
| Merge Sort           |        0.1440 |            6876 KB |
| Quick Sort           |        0.0630 |            6876 KB |
| Shell Sort           |        0.1430 |            6880 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |     2340.8380 |           20432 KB |
| Selection Sort       |      605.9530 |           18512 KB |
| Insertion Sort       |      744.6200 |           18588 KB |
| Merge Sort           |        0.1550 |           18660 KB |
| Quick Sort           |        0.1090 |           18552 KB |
| Shell Sort           |        0.3210 |           18556 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 1.000.000



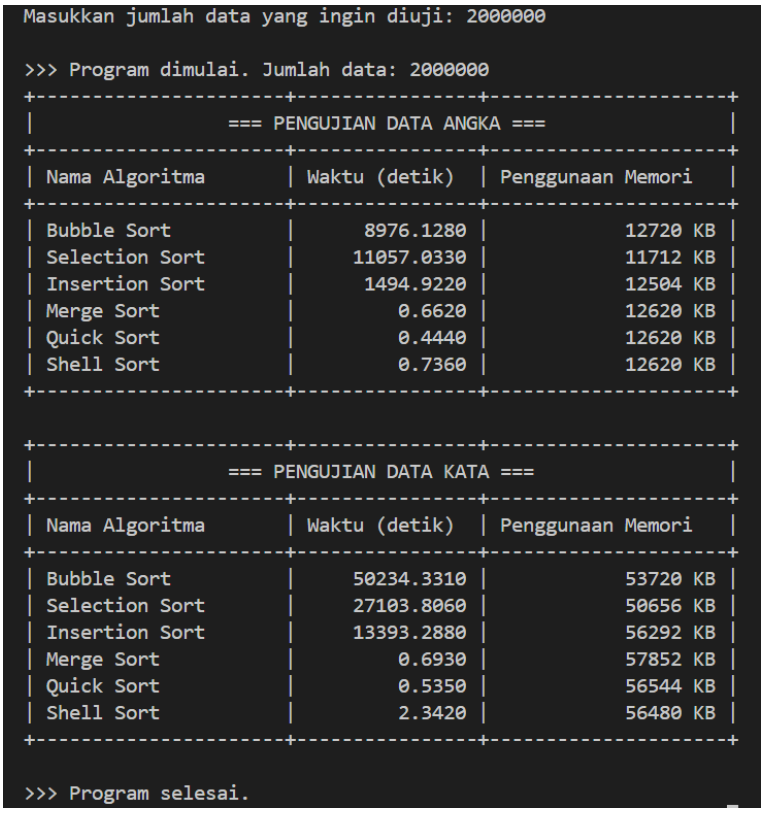
```
Masukkan jumlah data yang ingin diuji: 1000000

>>> Program dimulai. Jumlah data: 1000000
+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA ANGKA ===               |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |     4623.5420 |            7916 KB |
| Selection Sort       |      911.3410 |            8652 KB |
| Insertion Sort       |      247.3930 |            8624 KB |
| Merge Sort           |        0.2070 |            8704 KB |
| Quick Sort           |        0.1000 |            8704 KB |
| Shell Sort           |        0.2200 |            8704 KB |
+----------------------+---------------+--------------------+

+----------------------+---------------+--------------------+
|                === PENGUJIAN DATA KATA ===                |
+----------------------+---------------+--------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori  |
+----------------------+---------------+--------------------+
| Bubble Sort          |    15522.4960 |           29196 KB |
| Selection Sort       |     6750.9520 |           30176 KB |
| Insertion Sort       |     3661.0110 |           31696 KB |
| Merge Sort           |        0.3080 |           33128 KB |
| Quick Sort           |        0.2130 |           32176 KB |
| Shell Sort           |        0.7730 |           32140 KB |
+----------------------+---------------+--------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 1.500.000



```
Masukkan jumlah data yang ingin diuji: 1500000

>>> Program dimulai. Jumlah data: 1500000
+----------------------+---------------+---------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+---------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori   |
+----------------------+---------------+---------------------+
| Bubble Sort          |     4225.9220 |            8748 KB |
| Selection Sort       |     3496.7660 |           10540 KB |
| Insertion Sort       |      960.6760 |           10540 KB |
| Merge Sort           |        0.5430 |           10640 KB |
| Quick Sort           |        0.2820 |           10640 KB |
| Shell Sort           |        0.5900 |           10640 KB |
+----------------------+---------------+---------------------+

+----------------------+---------------+---------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+---------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori   |
+----------------------+---------------+---------------------+
| Bubble Sort          |    20930.2230 |           42656 KB |
| Selection Sort       |     7499.7960 |           45608 KB |
| Insertion Sort       |     8606.7710 |           45268 KB |
| Merge Sort           |        0.5220 |           46904 KB |
| Quick Sort           |        0.4090 |           45852 KB |
| Shell Sort           |        1.7410 |           45800 KB |
+----------------------+---------------+---------------------+

>>> Program selesai.
```

- Pengujian dengan jumlah data 2.000.000

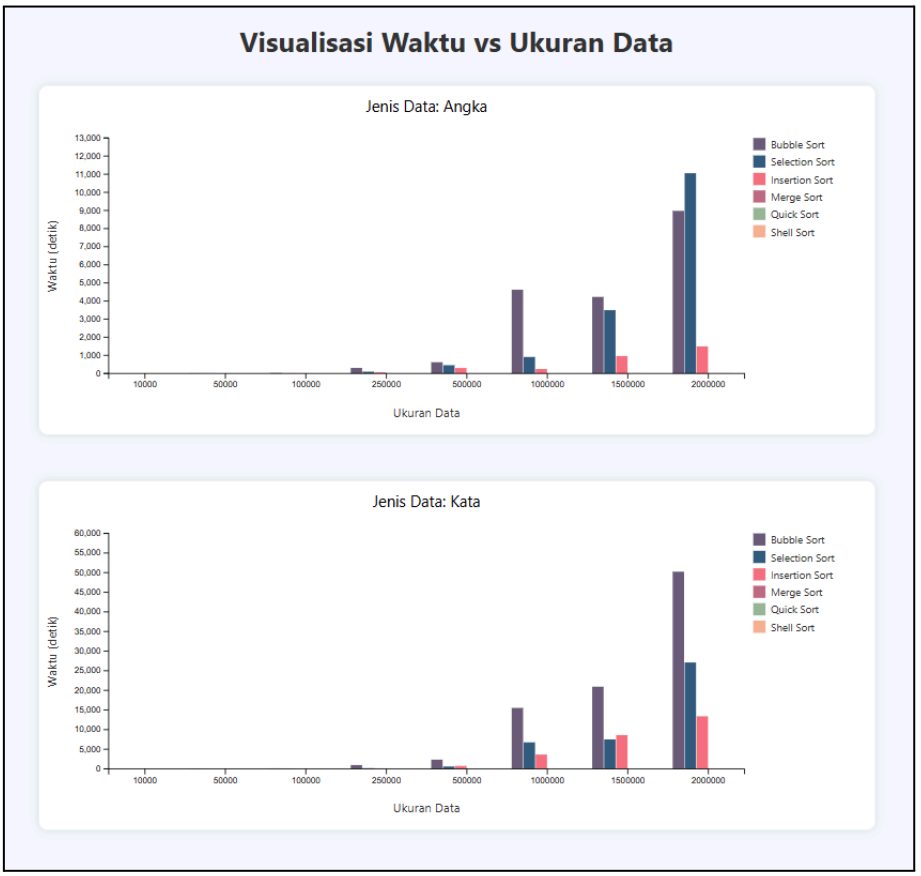


```
Masukkan jumlah data yang ingin diuji: 2000000

>>> Program dimulai. Jumlah data: 2000000
+----------------------+---------------+---------------------+
|                === PENGUJIAN DATA ANGKA ===                |
+----------------------+---------------+---------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori   |
+----------------------+---------------+---------------------+
| Bubble Sort          |     8976.1280 |           12720 KB |
| Selection Sort       |    11057.0330 |           11712 KB |
| Insertion Sort       |     1494.9220 |           12504 KB |
| Merge Sort           |        0.6620 |           12620 KB |
| Quick Sort           |        0.4440 |           12620 KB |
| Shell Sort           |        0.7360 |           12620 KB |
+----------------------+---------------+---------------------+

+----------------------+---------------+---------------------+
|                === PENGUJIAN DATA KATA ===                 |
+----------------------+---------------+---------------------+
| Nama Algoritma       | Waktu (detik) | Penggunaan Memori   |
+----------------------+---------------+---------------------+
| Bubble Sort          |    50234.3310 |           53720 KB |
| Selection Sort       |    27103.8060 |           50656 KB |
| Insertion Sort       |    13393.2880 |           56292 KB |
| Merge Sort           |        0.6930 |           57852 KB |
| Quick Sort           |        0.5350 |           56544 KB |
| Shell Sort           |        2.3420 |           56480 KB |
+----------------------+---------------+---------------------+

>>> Program selesai.
```

C. Grafik perbandingan waktu dan memori

Berikut adalah grafik perbandingan waktu dan memori yang digunakan dalam pengujian ini, saya akan menjelaskan perbandingannya berdasarkan jenis data yang digunakan yaitu data angka dan data kata.

- Grafik perbandingan waktu yang digunakan dan banyak data yang diuji

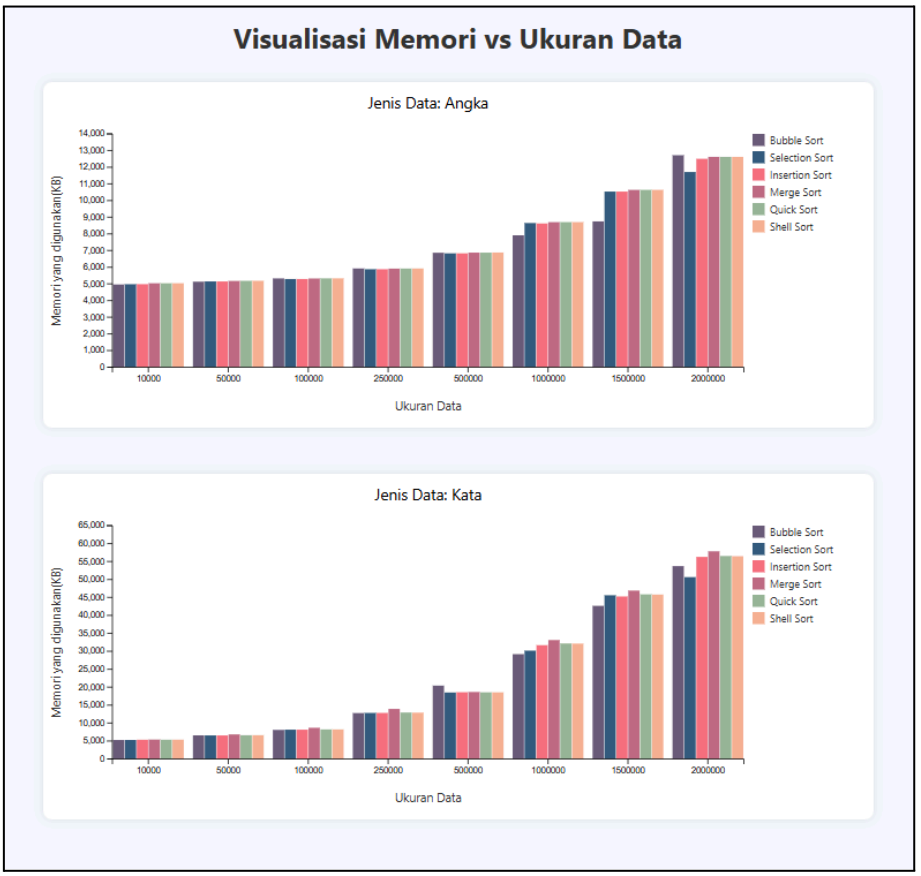


Dari grafik tersebut, dapat disimpulkan bahwa semua algoritma menunjukkan peningkatan waktu eksekusi seiring bertambahnya ukuran data, baik untuk jenis data angka maupun kata. Pada data angka, waktu eksekusi Quick Sort dan Merge Sort relatif lebih stabil dan lebih cepat dibandingkan algoritma lainnya. Sedangkan Bubble Sort, Selection Sort, dan Insertion Sort mengalami peningkatan waktu yang jauh lebih besar, terutama pada ukuran data yang sangat besar.

Pada data kata, peningkatan waktu eksekusi menjadi lebih tajam, dengan Bubble Sort dan Selection Sort menunjukkan waktu eksekusi yang paling tinggi. Merge Sort dan Quick Sort tetap menjadi algoritma yang paling efisien dari sisi waktu, sedangkan Shell Sort berada di posisi tengah, lebih baik daripada Bubble Sort, Selection Sort, dan Insertion Sort, tetapi masih kalah cepat dibandingkan Quick Sort dan Merge Sort.

Dengan demikian, untuk jenis data berukuran besar, baik angka maupun kata, Merge Sort dan Quick Sort adalah algoritma yang paling efisien dari sisi waktu eksekusi, sementara Bubble Sort, Selection Sort, dan Insertion Sort kurang disarankan karena memerlukan waktu eksekusi yang jauh lebih lama.

- Grafik perbandingan memori yang digunakan dan banyak data yang diuji



Dari grafik tersebut, dapat disimpulkan bahwa semua algoritma menunjukkan peningkatan penggunaan memori seiring bertambahnya ukuran data, baik untuk jenis data angka maupun kata. Pada data angka, penggunaan memori antar algoritma relatif seimbang, tanpa perbedaan yang signifikan. Namun, pada data kata, penggunaan memori meningkat lebih tajam, terutama pada ukuran data besar.

Bubble Sort dan Merge Sort cenderung menggunakan memori lebih tinggi pada data kata berukuran besar, sementara Shell Sort dan Selection Sort menunjukkan efisiensi memori yang lebih baik. Insertion Sort berada di tengah-tengah. Quick Sort tetap stabil dan cukup efisien dalam hal memori, meskipun tidak sehemat Shell Sort.

Dengan demikian, untuk jenis data kata dan ukuran besar, Shell Sort dan Selection Sort lebih efisien dari sisi memori, sementara Merge Sort dan Bubble Sort memerlukan perhatian lebih karena konsumsi memori yang tinggi.

D. Analisis dan kesimpulan

Berdasarkan hasil visualisasi perbandingan algoritma sorting terhadap ukuran data, baik dari segi waktu eksekusi maupun penggunaan memori, dapat ditarik beberapa kesimpulan penting.

Dari grafik waktu terhadap ukuran data, terlihat bahwa semua algoritma mengalami peningkatan waktu eksekusi seiring bertambahnya ukuran data, baik untuk jenis data angka maupun kata. Namun, Quick Sort dan Merge Sort menunjukkan performa yang paling efisien dan stabil dalam hal kecepatan, bahkan pada data berukuran besar. Sebaliknya, Bubble Sort, Selection Sort, dan Insertion Sort mengalami peningkatan waktu yang sangat signifikan dan cenderung tidak efisien saat menangani data dalam jumlah besar. Shell Sort berada di tengah-tengah, menunjukkan performa yang lebih baik dibanding tiga algoritma pertama, namun masih kalah cepat dibanding Quick Sort dan Merge Sort.

Sementara itu, dari grafik memori terhadap ukuran data, semua algoritma juga menunjukkan peningkatan penggunaan memori seiring bertambahnya ukuran data. Untuk data angka, penggunaan memori antar algoritma relatif seimbang, tanpa perbedaan yang mencolok. Namun pada data kata, perbedaan konsumsi memori menjadi lebih jelas, terutama saat ukuran data semakin besar. Bubble Sort dan Merge Sort terlihat menggunakan memori paling tinggi, sedangkan Shell Sort dan Selection Sort menunjukkan efisiensi memori yang lebih baik. Quick Sort tetap cukup stabil dan efisien dari sisi memori, meskipun tidak sehemat Shell Sort.

Dengan demikian, dapat disimpulkan bahwa:

- Quick Sort dan Merge Sort merupakan pilihan terbaik dari segi waktu eksekusi, terutama untuk data berukuran besar.
- Untuk efisiensi memori, Shell Sort dan Selection Sort menjadi alternatif yang lebih ringan, khususnya pada data berupa kata.
- Bubble Sort, Selection Sort, dan Insertion Sort secara umum kurang disarankan untuk data skala besar karena performa waktu dan/atau memori yang rendah.