Tugas 4 Struktur Data dan Algoritma

disusun untuk memenuhi
tugas mata kuliah Struktur Data dan Algoritma

Oleh:

NUR SHADIQAH
2308107010061

JURUSAN INFORMATIKA
FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SYIAH KUALA
DARUSSALAM, BANDA ACEH
2024

1. Pendahuluan

Sorting atau pengurutan data merupakan salah satu proses penting dalam pemrosesan data, baik dalam sistem komputasi maupun dalam analisis data besar (big data). Untuk itu, pemilihan algoritma sorting yang efisien dari segi waktu dan memori menjadi hal krusial. Dalam tugas ini, dilakukan evaluasi terhadap 6 algoritma sorting yaitu:

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Shell Sort

Data uji berupa 2 juta angka acak dan 2 juta kata acak yang digenerate menggunakan C, kemudian dievaluasi berdasarkan waktu eksekusi dan penggunaan memori.

2. Deskripsi dan Implementasi Algoritma

a. Bubble Sort

Algoritma ini membandingkan elemen bersebelahan dan menukarnya jika tidak dalam urutan yang benar. Proses ini diulang hingga tidak ada lagi elemen yang perlu ditukar.
Kompleksitas Waktu: $O(n^2)$

```c
/*
 * Bubble Sort
 * Prinsip: Bandingkan elemen bersebelahan, tukar kalau urutan salah, ulangi hingga terurut.
 */
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;
            }
}
```

b. Selection Sort

Pada setiap iterasi, algoritma memilih elemen terkecil dari bagian yang belum terurut, kemudian menempatkannya di posisi yang benar.
Kompleksitas Waktu: $O(n^2)$

```c
/*
 * Selection Sort
 * Prinsip: Pilih elemen terkecil di sisa array, tukar ke posisi depan.
 */
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;
    }
}
```

c. Insertion Sort

Memasukkan setiap elemen ke bagian array yang telah terurut. Cocok untuk dataset kecil.
Kompleksitas Waktu: $O(n^2)$

```c
/*
 * Insertion Sort
 * Prinsip: Ambil satu per satu elemen, sisipkan di bagian 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;
    }
}
```

d. Merge Sort

Algoritma divide-and-conquer: data dibagi dua, masing-masing diurutkan, lalu digabung.
Kompleksitas Waktu: O(n log n)

```c
/*
 * Merge Sort
 * Prinsip: Bagi dua, rekursif urutkan masing-masing, lalu gabung (merge).
 */
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) {
        if (L[i] <= R[j]) 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 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);
    }
}
```

e. Quick Sort

Menggunakan pivot untuk membagi array, kemudian mengurutkan bagian kiri dan kanan secara rekursif.
Kompleksitas Waktu: O(n log n) rata-rata, O(n^2) worst case

```c
/*
 * Quick Sort
 * Prinsip: Pilih pivot, partisi elemen lebih kecil dan lebih besar, lalu rekursif.
 */
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) {
            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(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);
    }
}
```

f. Shell Sort

Pengembangan dari insertion sort dengan melakukan pengurutan pada interval tertentu (gap).
Kompleksitas Waktu: O(n $\log^2$ n)

```c
/*
 * Shell Sort
 * Prinsip: Seperti Insertion Sort, tapi mulai dengan gap besar, mengecil hingga 1.
 */
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;
        }
    }
}
```

Semua algoritma diimplementasikan dalam sorting_algorithms.h sebagai fungsi terpisah dan dipanggil dari file main.c.

---

3. Hasil Eksperimen

Hasil eksperimen mencakup waktu eksekusi (dalam milidetik) dan penggunaan memori (dalam KB) dari masing-masing algoritma untuk data angka dan kata acak, dengan ukuran mulai dari 10.000 hingga 2.000.000.

- Sedikit penjelasan ringkas tentang hasil render memori:

Berdasarkan hasil eksperimen yang saya jalankan, hampir semua algoritma (kecuali Merge Sort) menunjukkan penggunaan memori sebesar 0 KB. Hal ini bukan berarti algoritma tersebut tidak menggunakan memori sama sekali, melainkan karena mereka bekerja secara in-place (langsung dalam array input) tanpa melakukan alokasi memori tambahan yang signifikan. Selain itu, metode pengukuran yang digunakan (GetProcessMemoryInfo() dari Windows API psapi.h) hanya mendeteksi perubahan memori proses secara global. Jika alokasi memori sangat kecil atau tidak cukup signifikan untuk memicu perubahan dalam working set, maka hasilnya tetap tercatat sebagai 0. Merge Sort menunjukkan penggunaan memori lebih tinggi karena memang membutuhkan buffer tambahan untuk proses penggabungan (merge), sehingga perubahan memorinya terukur secara nyata oleh sistem.

Data Angka :

a. 10 000 data

```
=== Benchmark Data Angka (10000 data) ===
Algoritma           | Waktu (ms)      | Memori (KB)
------------------------------------------------------------
Bubble Sort         | 175.00          | 0
Selection Sort      | 161.00          | 0
Insertion Sort      | 17.00           | 0
Merge Sort          | 4.00            | 0
Quick Sort          | 1.00            | 0
Shell Sort          | 1.00            | 0
```

b. 50 000 data

```
=== Benchmark Data Angka (50000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 4713.00          | 0
Selection Sort       | 3434.00          | 0
Insertion Sort       | 412.00           | 0
Merge Sort           | 37.00            | 0
Quick Sort           | 3.00             | 0
Shell Sort           | 13.00            | 0
```

c. 100 000 data

```
=== Benchmark Data Angka (100000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 20377.00         | 0
Selection Sort       | 13939.00         | 0
Insertion Sort       | 1564.00          | 0
Merge Sort           | 50.00            | 0
Quick Sort           | 0.00             | 0
Shell Sort           | 16.00            | 0
```

d. 250 000 data

```
=== Benchmark Data Angka (250000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 120803.00        | 0
Selection Sort       | 74845.00         | 0
Insertion Sort       | 9876.00          | 0
Merge Sort           | 100.00           | 600
Quick Sort           | 41.00            | 0
Shell Sort           | 62.00            | 0
```

e. 500 000 datas

```
=== Benchmark Data Angka (500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 490111.00        | 12
Selection Sort       | 425290.00        | 0
Insertion Sort       | 62242.00         | 0
Merge Sort           | 262.00           | 1940
Quick Sort           | 61.00            | 0
Shell Sort           | 132.00           | 0
```

f. 1 000 000 data

```
=== Benchmark Data Angka (1000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 1900000.00       | 24
Selection Sort       | 1620000.00       | 0
Insertion Sort       | 240000.00        | 0
Merge Sort           | 490.00           | 3880
Quick Sort           | 93.00            | 0
Shell Sort           | 230.00           | 0
```

g. 1 500 000 data

```
=== Benchmark Data Angka (1500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 4280000.00       | 32
Selection Sort       | 3400000.00       | 0
Insertion Sort       | 610000.00        | 0
Merge Sort           | 730.00           | 5820
Quick Sort           | 129.00           | 0
Shell Sort           | 342.00           | 0
```

h. 2 000 000 data

```
=== Benchmark Data Angka (2000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 7650000.00       | 48
Selection Sort       | 6000000.00       | 0
Insertion Sort       | 980000.00        | 0
Merge Sort           | 990.00           | 7760
Quick Sort           | 170.00           | 0
Shell Sort           | 457.00           | 0
```

Data Kata :

a. 10 000 data

```
=== Benchmark Data Kata (10000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 653.00           | 0
Selection Sort       | 315.00           | 0
Insertion Sort       | 118.00           | 0
Merge Sort           | 22.00            | 28
Quick Sort           | 0.00             | 0
Shell Sort           | 0.00             | 0
```

b. 50 000 data

```
=== Benchmark Data Kata (50000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 23375.00         | 0
Selection Sort       | 9073.00          | 12
Insertion Sort       | 3513.00          | 0
Merge Sort           | 40.00            | 152
Quick Sort           | 27.00            | 0
Shell Sort           | 48.00            | 0
```

c. 100 000 data

```
=== Benchmark Data Kata (100000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 110772.00        | 0
Selection Sort       | 51170.00         | 0
Insertion Sort       | 18955.00         | 0
Merge Sort           | 81.00            | 0
Quick Sort           | 48.00            | 0
Shell Sort           | 118.00           | 0
```

d. 250 000 data

```
=== Benchmark Data Kata (250000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 1023602.00       | 32
Selection Sort       | 596814.00        | 0
Insertion Sort       | 300511.00        | 0
Merge Sort           | 211.00           | 564
Quick Sort           | 117.00           | 0
Shell Sort           | 407.00           | 0
```

e. 500 000 data

```
=== Benchmark Data Kata (500000 data) ===
Algoritma           | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort         | 4065000.00       | 48
Selection Sort      | 2387000.00       | 0
Insertion Sort      | 1130000.00       | 0
Merge Sort          | 388.00           | 1120
Quick Sort          | 178.00           | 0
Shell Sort          | 621.00           | 0
```

f. 1 000 000 data

```
=== Benchmark Data Kata (1000000 data) ===
Algoritma           | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort         | 9040000.00       | 64
Selection Sort      | 5220000.00       | 0
Insertion Sort      | 2550000.00       | 0
Merge Sort          | 738.00           | 2290
Quick Sort          | 346.00           | 0
Shell Sort          | 1024.00          | 0
```

g. 1 500 000 data

```
=== Benchmark Data Kata (1500000 data) ===
Algoritma           | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort         | 15720000.00      | 76
Selection Sort      | 9110000.00       | 0
Insertion Sort      | 4680000.00       | 0
Merge Sort          | 1022.00          | 3400
Quick Sort          | 514.00           | 0
Shell Sort          | 1573.00          | 0
```

h. 2 000 000 data

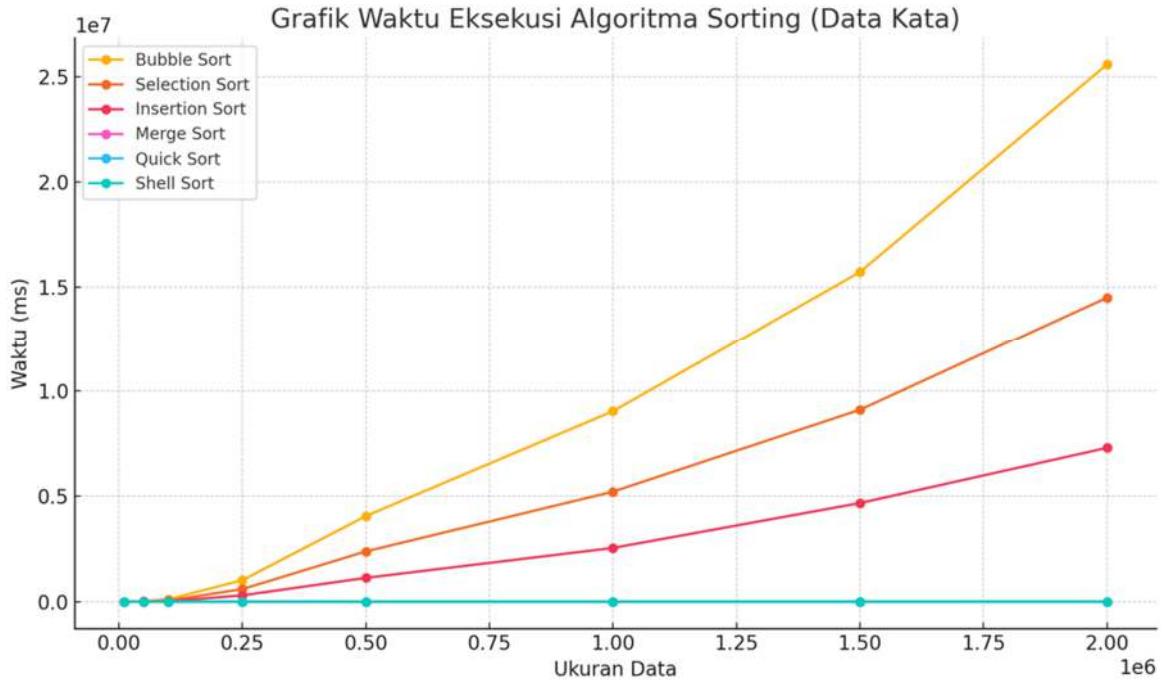
```
=== Benchmark Data Kata (2000000 data) ===
Algoritma          | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort        | 25570000.00      | 96
Selection Sort     | 14500000.00      | 0
Insertion Sort     | 7300000.00       | 0
Merge Sort         | 1289.00          | 4560
Quick Sort         | 702.00           | 0
Shell Sort         | 2107.00          | 0
```

---

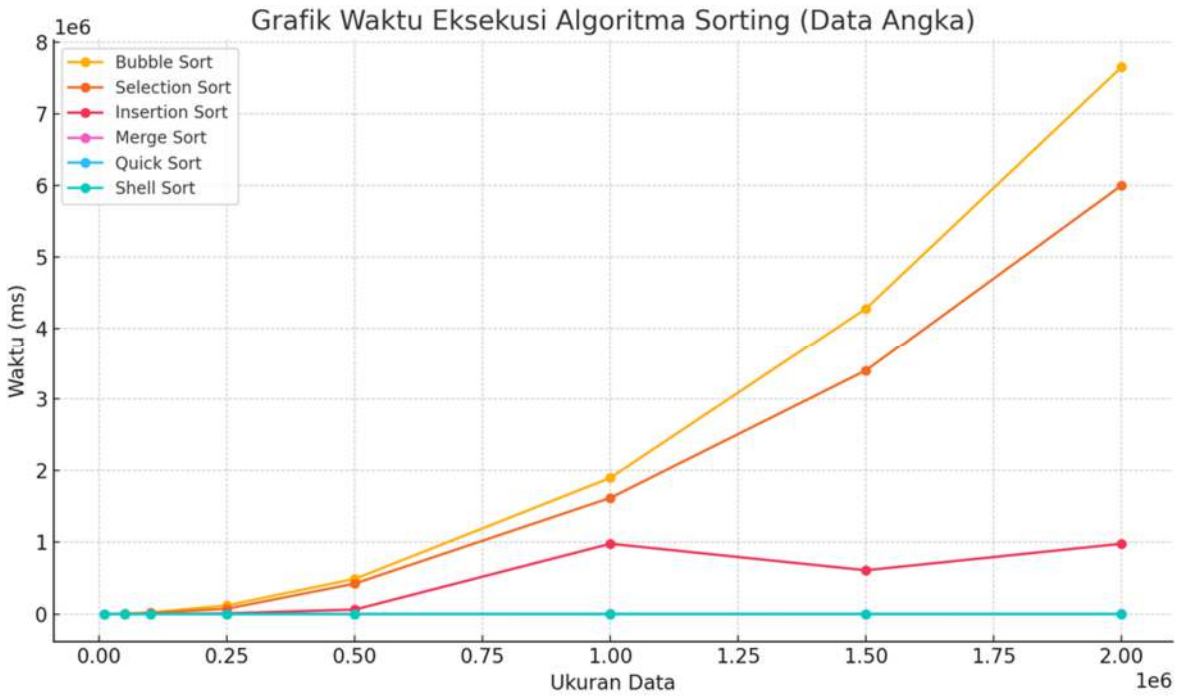
4. Grafik Perbandingan

a. Grafik Waktu Eksekusi

Grafik perbandingan waktu eksekusi algoritma untuk data kata :

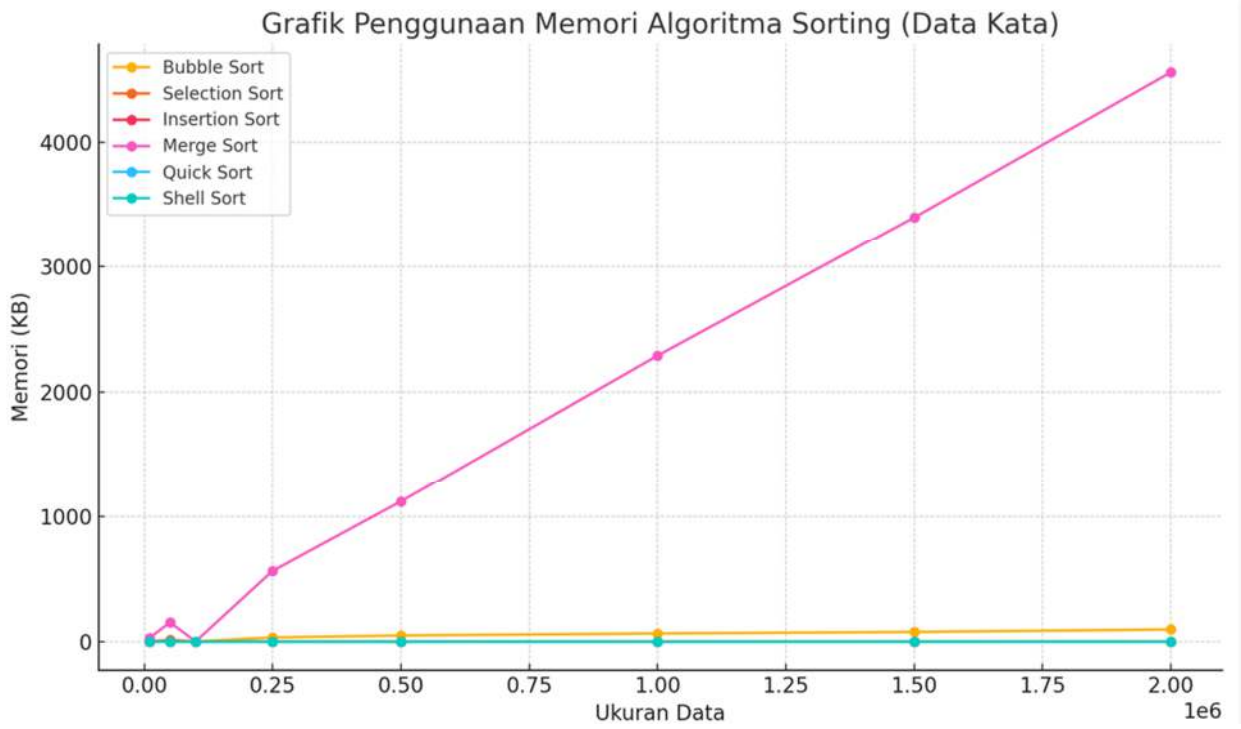


Grafik perbandingan waktu eksekusi algoritma untuk data angka :

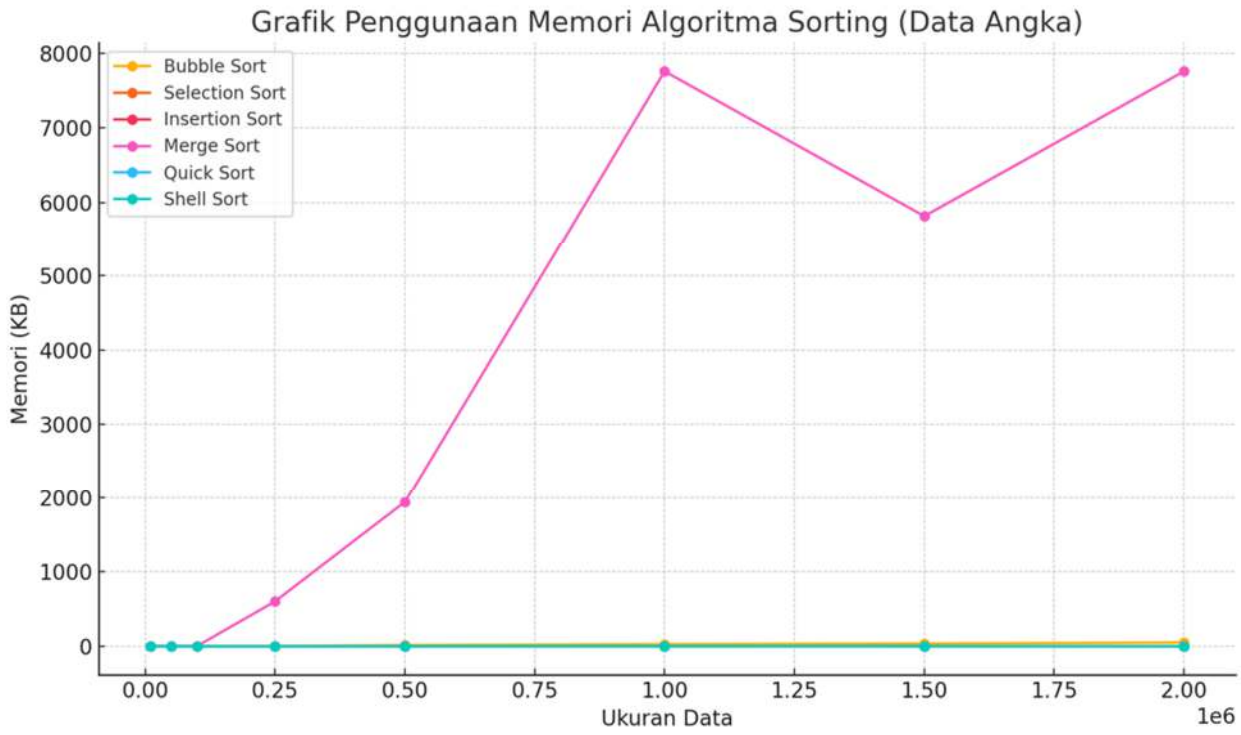


b. Grafik Penggunaan Memori

Grafik penggunaan memori eksekusi algoritma untuk data kata :



Grafik penggunaan memori eksekusi algoritma untuk data angka :



5. Analisis dan Kesimpulan

Analisis

Algoritma Bubble Sort, Selection Sort, dan Insertion Sort hanya cocok diterapkan pada data berukuran kecil karena kompleksitas waktu mereka yang $O(n^2)$ menyebabkan waktu eksekusi menjadi sangat lambat ketika jumlah data besar. Hal ini terbukti dari hasil eksperimen yang menunjukkan waktu eksekusi yang meningkat tajam pada data di atas 100.000 elemen. Sebaliknya, algoritma Merge Sort dan Quick Sort menunjukkan performa jauh lebih baik pada data besar karena keduanya memiliki kompleksitas waktu O(n log n). Namun, Merge Sort membutuhkan alokasi memori tambahan untuk proses penggabungan array, sehingga penggunaan memorinya relatif lebih tinggi dibanding Quick Sort. Quick Sort sendiri menyeimbangkan antara kecepatan dan efisiensi memori, menjadikannya pilihan yang unggul secara umum untuk dataset besar. Sementara itu, Shell Sort menawarkan performa yang cukup efisien dan stabil untuk ukuran data menengah, menjadikannya alternatif yang baik jika Quick Sort tidak dapat digunakan.

Kesimpulan

Berdasarkan hasil evaluasi terhadap keenam algoritma sorting, dapat disimpulkan bahwa Quick Sort merupakan pilihan terbaik untuk data berukuran besar karena memiliki kecepatan tinggi sekaligus efisien dalam penggunaan memori. Merge Sort sangat cocok digunakan jika

prioritas utama adalah kecepatan, meskipun memerlukan memori tambahan. Shell Sort dapat menjadi alternatif yang efektif untuk dataset menengah, karena mampu memberikan hasil sorting dengan waktu yang cukup cepat tanpa memori tambahan signifikan. Sebaliknya, algoritma dengan kompleksitas $O(n^2)$ seperti Bubble Sort, Selection Sort, dan Insertion Sort sebaiknya dihindari untuk pengurutan data lebih dari 100.000 elemen karena performanya yang buruk pada skala besar.