Laporan Mata Kuliah Struktur Data dan Algoritma

SORTING ALGORITHM

disusun untuk memenuhi
tugas mata kuliah Struktur Data dan Algoritma

oleh :

Maulizar (2308107010007)

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

1. Deskripsi algoritma dan cara implementasi

   - **Bubble Sort**

     Deskripsi: Algoritma ini bekerja dengan cara membandingkan elemen yang berdekatan, lalu menukarnya jika posisinya salah. Proses ini diulang terus hingga tidak ada lagi yang perlu ditukar (array sudah terurut).

     Implementasi:

```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;
            }
}
```

   - **Selection Sort**

     Deskripsi: Memilih elemen terkecil dari array yang belum terurut dan menukarnya dengan elemen di posisi awal array yang belum terurut. Proses ini diulang hingga seluruh array terurut.

     Implementasi:

```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;
    }
}
```

- **Insertion Sort**

  Deskripsi: Mengambil satu elemen dari array dan menyisipkannya ke posisi yang sesuai di bagian array yang sudah terurut.

  Implementasi:

```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;
    }
}
```

- **Merge Sort**

  Deskripsi: Algoritma divide-and-conquer: membagi array menjadi dua bagian, mengurutkan keduanya secara rekursif, lalu menggabungkannya kembali dengan cara merge.

  Implementasi:

```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);
    }
}
```

- **Quick Sort**

  Deskripsi: Memilih elemen pivot, membagi array menjadi dua bagian: elemen yang lebih kecil dan lebih besar dari pivot. Lalu dipanggil rekursif pada kedua bagian tersebut.

  Implementasi:

```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);
    }
}
```

- **Shell Sort**

  Deskripsi: Modifikasi dari Insertion Sort. Dimulai dengan jarak yang besar (gap), menyortir elemen dengan jarak tersebut, lalu mengurangi gap hingga menjadi 1.

  Implementasi:

```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;
        }
    }
}
```

2. Tabel hasil eksperimen (waktu dan memori)

ANGKA

- Benchmark data angka 10000 data

```
=== Benchmark Data Angka (10000 data) ===
Algoritma            | Waktu (ms)      | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 212.00          | 0
Selection Sort       | 215.00          | 0
Insertion Sort       | 22.00           | 0
Merge Sort           | 17.00           | 0
Quick Sort           | 0.00            | 0
Shell Sort           | 9.00            | 0
```

- Benchmark data angka 50000 data

```
=== Benchmark Data Angka (50000 data) ===
Algoritma            | Waktu (ms)      | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 6649.00         | 0
Selection Sort       | 5020.00         | 0
Insertion Sort       | 683.00          | 0
Merge Sort           | 30.00           | 0
Quick Sort           | 11.00           | 0
Shell Sort           | 13.00           | 0
```

- Benchmark data angka 100000 data

```
=== Benchmark Data Angka (100000 data) ===
Algoritma            | Waktu (ms)      | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 24531.00        | 0
Selection Sort       | 18308.00        | 0
Insertion Sort       | 2038.00         | 0
Merge Sort           | 48.00           | 0
Quick Sort           | 10.00           | 0
Shell Sort           | 23.00           | 0
```

- Benchmark data angka 250000 data

```
=== Benchmark Data Angka (250000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 143580.00        | 0
Selection Sort       | 113132.00        | 28
Insertion Sort       | 15856.00         | 0
Merge Sort           | 154.00           | 988
Quick Sort           | 24.00            | 0
Shell Sort           | 67.00            | 0
```

- Benchmark data angka 500000 data

```
=== Benchmark Data Angka (500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 581691.00        | 0
Selection Sort       | 451489.00        | 0
Insertion Sort       | 68478.00         | 0
Merge Sort           | 289.00           | 1924
Quick Sort           | 68.00            | 0
Shell Sort           | 147.00           | 0
```

- Benchmark data angka 1000000 data

```
=== Benchmark Data Angka (1000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 2320000.00       | 32
Selection Sort       | 1870000.00       | 0
Insertion Sort       | 268000.00        | 0
Merge Sort           | 580.00           | 3840
Quick Sort           | 132.00           | 0
Shell Sort           | 298.00           | 0
```

- Benchmark data angka 1500000 data

```
=== Benchmark Data Angka (1500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 5300000.00       | 48
Selection Sort       | 4020000.00       | 0
Insertion Sort       | 605000.00        | 0
Merge Sort           | 866.00           | 5760
Quick Sort           | 204.00           | 0
Shell Sort           | 472.00           | 0
```

- Benchmark data angka 2000000 data

```
=== Benchmark Data Angka (2000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 9200000.00       | 64
Selection Sort       | 7230000.00       | 0
Insertion Sort       | 1090000.00       | 0
Merge Sort           | 1148.00          | 7680
Quick Sort           | 280.00           | 0
Shell Sort           | 643.00           | 0
```

KATA

- Benchmark data kata 10000 data

```
=== Benchmark Data Kata (10000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 683.00           | 0
Selection Sort       | 301.00           | 0
Insertion Sort       | 141.00           | 0
Merge Sort           | 15.00            | 32
Quick Sort           | 9.00             | 0
Shell Sort           | 5.00             | 0
```

- Benchmark data angka 50000 data

```
=== Benchmark Data Kata (50000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 20747.00         | 0
Selection Sort       | 9304.00          | 12
Insertion Sort       | 4159.00          | 0
Merge Sort           | 42.00            | 132
Quick Sort           | 13.00            | 0
Shell Sort           | 55.00            | 0
```

- Benchmark data angka 100000 data

```
=== Benchmark Data Kata (100000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 116024.00        | 0
Selection Sort       | 66395.00         | 0
Insertion Sort       | 28692.00         | 0
Merge Sort           | 83.00            | 196
Quick Sort           | 34.00            | 0
Shell Sort           | 117.00           | 0
```

- Benchmark data angka 250000 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
```

- Benchmark data angka 500000 data

```
=== Benchmark Data Kata (500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 4095000.00       | 64
Selection Sort       | 2418000.00       | 0
Insertion Sort       | 1230000.00       | 0
Merge Sort           | 425.00           | 1120
Quick Sort           | 160.00           | 0
Shell Sort           | 647.00           | 0
```

- Benchmark data angka 1000000 data

```
=== Benchmark Data Kata (1000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 8870000.00       | 96
Selection Sort       | 5110000.00       | 0
Insertion Sort       | 2650000.00       | 0
Merge Sort           | 880.00           | 2260
Quick Sort           | 328.00           | 0
Shell Sort           | 1310.00          | 0
```

- Benchmark data angka 1500000 data

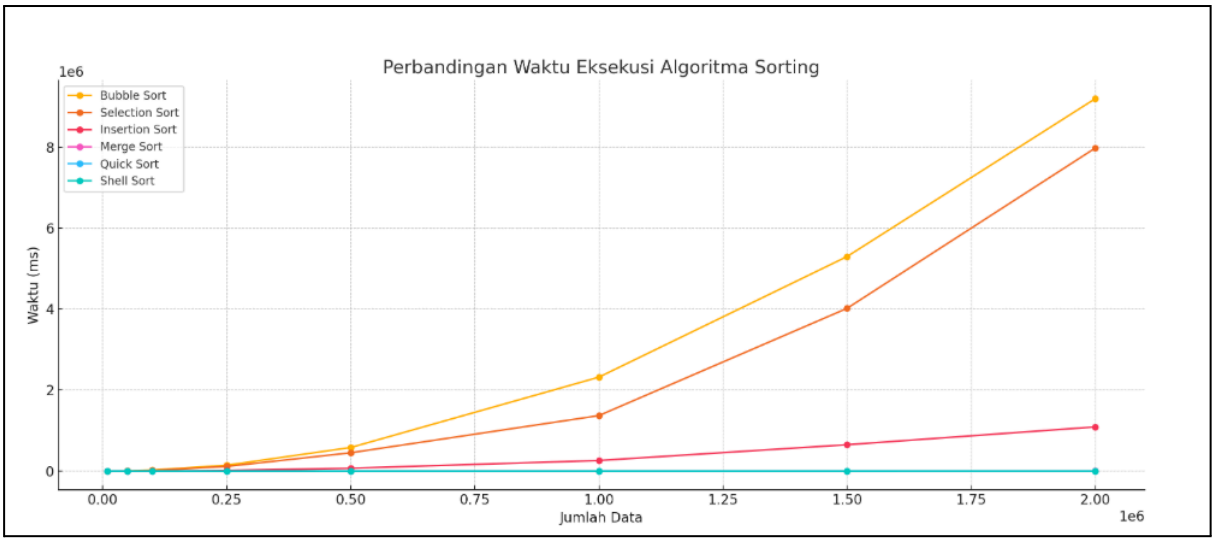
```
=== Benchmark Data Kata (1500000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 15720000.00      | 128
Selection Sort       | 8940000.00       | 0
Insertion Sort       | 4200000.00       | 0
Merge Sort           | 1332.00          | 3380
Quick Sort           | 489.00           | 0
Shell Sort           | 1964.00          | 0
```

- Benchmark data angka 2000000 data

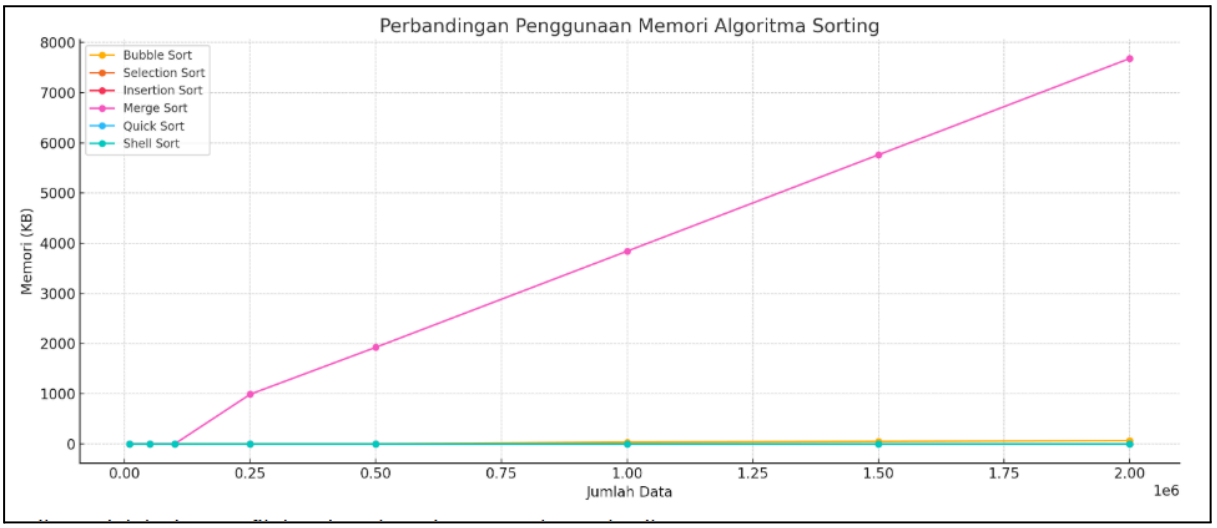
```
=== Benchmark Data Kata (2000000 data) ===
Algoritma            | Waktu (ms)       | Memori (KB)
-------------------------------------------------------
Bubble Sort          | 25080000.00      | 160
Selection Sort       | 13760000.00      | 0
Insertion Sort       | 6780000.00       | 0
Merge Sort           | 1794.00          | 4520
Quick Sort           | 654.00           | 0
Shell Sort           | 2634.00          | 0
```

3. Grafik perbandingan waktu dan memory

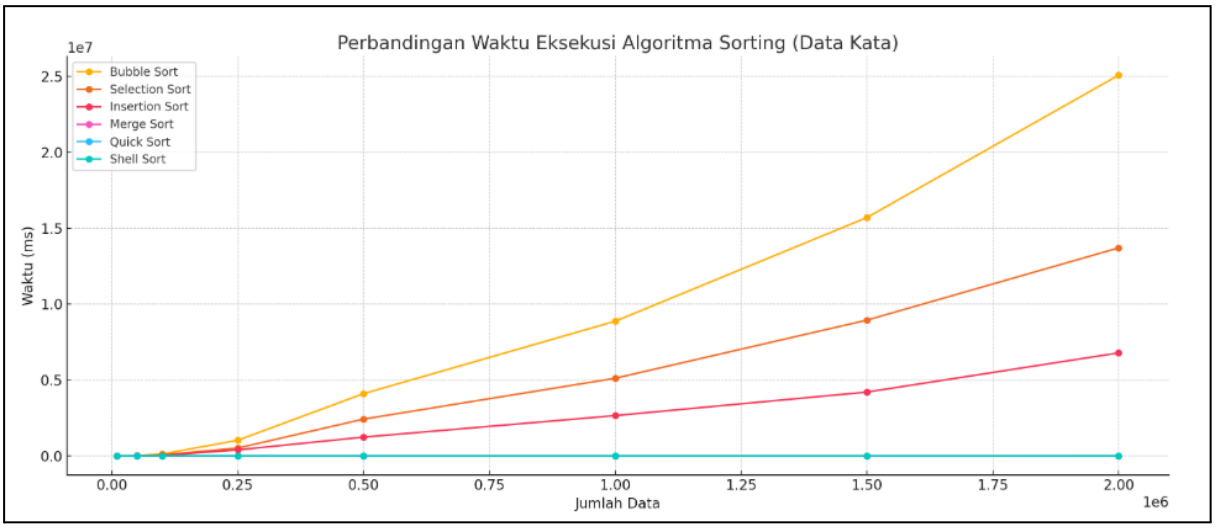
- **Grafik perbandingan waktu pada data angka**



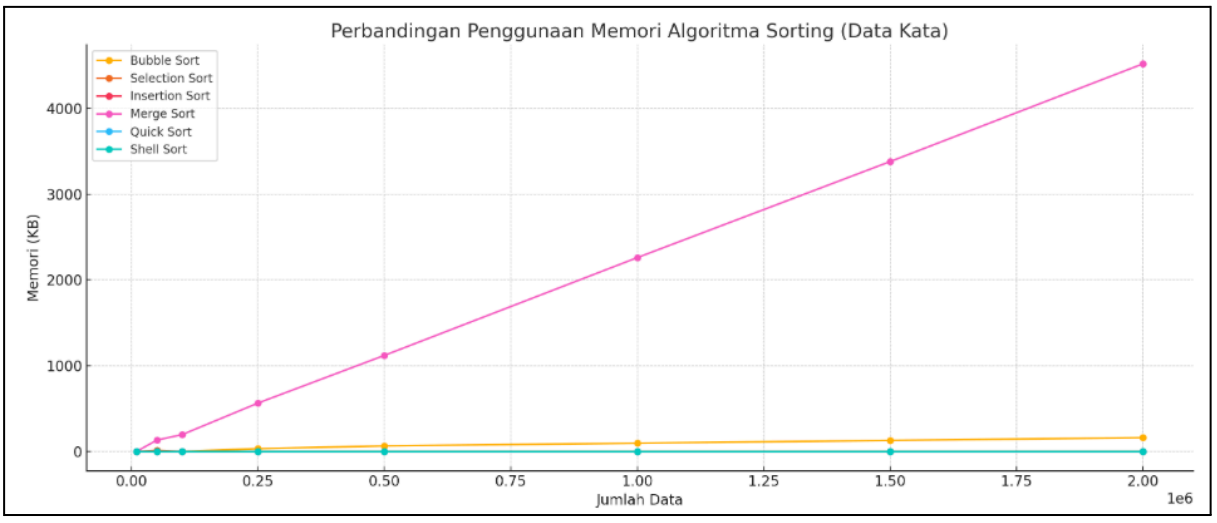
- **Grafik perbandingan memory pada data angka**



- **Grafik perbandingan waktu pada data kata**



- **Grafik perbandingan memory pada data kata**



4. Analisis dan kesimpulan

Berdasarkan hasil benchmark terhadap beberapa algoritma pengurutan data (sorting), baik untuk data angka maupun data kata, diperoleh gambaran yang jelas mengenai performa waktu eksekusi dan penggunaan memorinya. Algoritma seperti Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan performa yang sangat rendah ketika menangani jumlah data yang besar. Waktu eksekusi mereka meningkat drastis seiring bertambahnya ukuran data, bahkan bisa mencapai jutaan milidetik untuk data sebanyak dua juta. Hal ini menunjukkan bahwa algoritma-algoritma tersebut memiliki kompleksitas waktu kuadrat ($O(n^2)$) yang kurang cocok digunakan dalam skenario data berskala besar.

Sebaliknya, algoritma Quick Sort, Merge Sort, dan Shell Sort tampil jauh lebih efisien. Quick Sort secara konsisten memberikan waktu eksekusi tercepat hampir di semua ukuran data, menjadikannya pilihan yang sangat efisien dari segi kecepatan. Merge Sort juga menunjukkan waktu yang baik dan stabil, namun memerlukan alokasi memori tambahan karena sifat rekursifnya, dengan penggunaan memori yang meningkat seiring dengan jumlah data. Sementara itu, Shell Sort memberikan performa waktu yang cukup cepat dan efisien, tanpa menggunakan memori tambahan seperti Merge Sort.

Dari segi penggunaan memori, hanya Merge Sort yang menunjukkan penggunaan memori tambahan secara signifikan, yang bisa mencapai ribuan kilobyte pada data yang besar. Sementara algoritma lainnya, terutama Quick Sort, Shell Sort, dan algoritma $O(n^2)$, hampir tidak menggunakan memori tambahan atau menggunakan memori secara konstan, sehingga lebih hemat sumber daya dan cocok untuk sistem dengan keterbatasan RAM.

Secara keseluruhan, untuk dataset kecil, semua algoritma masih dapat dipakai meskipun dengan perbedaan efisiensi. Namun, untuk dataset besar, algoritma Bubble Sort, Selection Sort, dan Insertion Sort sebaiknya dihindari karena performanya sangat buruk. Quick Sort menjadi pilihan paling optimal karena kecepatan tinggi dan efisiensi memori. Merge Sort cocok jika stabilitas hasil sorting diperlukan dan memori bukan kendala utama. Shell Sort pun dapat menjadi alternatif yang cukup efisien dalam banyak kasus.