Laporan Mata Kuliah Struktur Data dan Algoritma

Perbandingan Algoritma Sorting

disusun untuk memenuhi tugas
Mata Kuliah Struktur Data dan Algoritma

Oleh:

SHANIA RIZKA ANINDIA
2308107010067

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

1. Pendahuluan

Pengurutan (sorting) merupakan salah satu operasi dasar dalam ilmu komputer yang sangat penting dalam pengolahan data. Tujuan utama dari sorting adalah menyusun elemen-elemen dalam suatu struktur data ke dalam urutan tertentu, baik menaik (ascending) maupun menurun (descending), guna mempermudah proses analisis dan manipulasi data..

Terdapat berbagai jenis algoritma sorting, masing-masing dengan strategi dan kompleksitas yang berbeda. bubble sort**,** selection sort**,** dan insertion sort merupakan algoritma sederhana yang mudah dipahami, namun memiliki keterbatasan efisiensi terutama pada data berskala besar karena kompleksitas waktu $O(n^2)$. Sebaliknya, algoritma seperti merge sort dan quick sort mengusung pendekatan divide and conquer yang lebih efisien dengan kompleksitas rata-rata O(n log n). Sementara itu, Shell Sort hadir sebagai optimasi dari Insertion Sort dengan pendekatan berbasis gap, menawarkan performa yang lebih baik pada data besar.

Perbandingan antara algoritma-algoritma ini tidak hanya mencakup kecepatan dalam menyelesaikan proses pengurutan, tetapi juga efisiensi penggunaan sumber daya seperti memori. Oleh karena itu, analisis terhadap waktu eksekusi dan konsumsi memori dari masing-masing algoritma menjadi penting untuk menentukan penggunaannya dalam konteks nyata.

2. Deskripsi Algoritma dan cara implementasi

a. Bubble Sort

Bubble Sort adalah algoritma pengurutan sederhana yang bekerja dengan cara membandingkan pasangan elemen yang berdekatan**,** dan menukarnya jika urutannya salah**.** Proses ini dilakukan secara berulang-ulang sampai tidak ada lagi elemen yang perlu ditukar, yang berarti array sudah dalam keadaan terurut. Implementasi algoritma ini menggunakan dua loop bersarang, di mana loop luar mengontrol jumlah iterasi dan loop dalam membandingkan serta menukar elemen. Untuk pengurutan data bertipe string, perbandingan dilakukan menggunakan fungsi strcmp() untuk membandingkan secara leksikografis,

```c
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        int swapped = 0;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = 1;
            }
        }
        if (!swapped) break;
    }
}
```

Gambar 1. Bubble Sort

b. Selection Sort

Selection Sort adalah algoritma pengurutan yang bekerja dengan cara mencari elemen terkecil dari bagian array yang belum terurut dan menukarnya dengan elemen pertama dari bagian tersebut. Kemudian, pencarian diulang untuk elemen kedua terkecil, dan seterusnya, hingga seluruh array terurut. Implementasi algoritma ini menggunakan dua loop. Loop pertama bertugas menandai posisi saat ini, dan loop kedua bertugas mencari elemen minimum di sisa array. Untuk string, perbandingan juga menggunakan strcmp() untuk menentukan urutan berdasarkan abjad.

```c
void selectionSort(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;
            }
        }
        if (minIdx != i) {
            int temp = arr[i];
            arr[i] = arr[minIdx];
            arr[minIdx] = temp;
        }
    }
}
```

Gambar 2. Selection Sort

c. Insertion Sort

Insertion Sort melakukan pengurutan dengan menyusun array dengan membangun bagian yang sudah terurut satu per satu. Untuk setiap elemen, algoritma akan menyisipkannya ke posisi yang tepat dalam bagian yang sudah terurut sebelumnya. Proses ini dilakukan dengan cara menggeser elemen yang lebih besar ke kanan hingga ditemukan posisi yang tepat. Implementasi untuk data numerik cukup efisien untuk data kecil atau hampir terurut. Sedangkan untuk string, proses penyisipan dilakukan berdasarkan perbandingan leksikografis antar kata.

```
void insertionSort(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;
    }
}
```

Gambar 3. Insertion Sort

d. Merge Sort

Merge Sort adalah algoritma divide and conquer yang membagi array menjadi dua bagian, mengurutkan masing-masing bagian secara rekursif, lalu menggabungkan kembali dua bagian tersebut menjadi satu array yang terurut. Proses penggabungan dilakukan dengan membandingkan elemen-elemen dari kedua sub-array dan menyusunnya ke array utama secara bertahap. Dalam implementasinya, array sementara digunakan untuk menampung hasil penggabungan sementara. Untuk string, penggabungan dilakukan dengan membandingkan elemen menggunakan strcmp().

```c
void merge(int arr[], int l, int m, int r) {
    int n1 = m - l + 1;
    int n2 = r - m;

    int *L = (int *)malloc(n1 * sizeof(int));
    int *R = (int *)malloc(n2 * sizeof(int));
    if (L == NULL || R == NULL) {
        printf("Gagal mengalokasikan memori.\n");
        exit(EXIT_FAILURE);
    }

    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);
}
```

Gambar 4. Merge Sort

e. Quick sort

Quick Sort juga menggunakan pendekatan divide and conquer dengan memilih satu elemen sebagai pivot, lalu membagi array menjadi dua bagian, satu berisi elemen yang lebih kecil dari pivot, dan satu lagi yang lebih besar. Setelah itu, algoritma memanggil dirinya sendiri secara rekursif pada kedua bagian tersebut. Implementasi untuk data angka menggunakan logika partisi berdasarkan nilai, sedangkan untuk string menggunakan perbandingan leksikografis dengan strcmp() untuk membagi array berdasarkan urutan abjad.

```c
int partition(int arr[], int low, int high) {
    int pivot = arr[high]; // elemen terakhir sebagai pivot
    int i = low - 1;

    for (int j = low; j < high; j++) {
        if (arr[j] < pivot) {
            i++;
            int temp = arr[i];
            arr[i] = arr[j];
            arr[j] = temp;
        }
    }
    int temp = arr[i + 1];
    arr[i + 1] = arr[high];
    arr[high] = temp;

    return i + 1;
}

// Fungsi utama Quick Sort untuk angka
void quickSort(int arr[], int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high); // partisi
        quickSort(arr, low, pi - 1);        // kiri
        quickSort(arr, pi + 1, high);       // kanan
    }
}
```

Gambar 5. Quick Sort

f. Shell Sort

Shell Sort adalah versi optimalisasi dari Insertion Sort yang menggunakan gap (jarak) antar elemen yang dibandingkan. Mula-mula, elemen yang berjauhan dibandingkan dan disisipkan, lalu jarak dikurangi secara bertahap hingga menjadi 1, pada tahap ini menjadi mirip dengan Insertion Sort biasa. Hal ini membuat proses penyisipan lebih efisien dibandingkan Insertion Sort klasik. Dalam implementasinya, gap dikurangi secara sistematis, dan perbandingan string dilakukan menggunakan strcmp().

```
void shellSort(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;
            // Sisipkan temp ke posisi yang tepat dalam gap
            for (j = i; j >= gap && arr[j - gap] > temp; j -= gap) {
                arr[j] = arr[j - gap];
            }
            arr[j] = temp;
        }
    }
}
```

Gambar 6. Shell Sort

3. Tabel Hasil Eksperimen (waktu dan memori)

a. Data angka

Jumlah data	Algoritma	Waktu (s)	Memory (MB)
10.000	Bubble sort	0,57	99,71
	Selection sort	0,26	99,71
	Insertion sort	0,13	99,71
	Merge sort	0,01	99,77
	Quick sort	0,00	99,77
	Shell sort	0,01	99,77
50.000	Bubble sort	14,99	99,92
	Selection sort	6,28	99,88
	Insertion sort	3,50	99,88
	Merge sort	0,04	99,86
	Quick sort	0,02	00,86
	Shell sort	0,03	99,86
100.000	Bubble sort	60,22	99,68
	Selection sort	24,60	99,68
	Insertion sort	12,93	99,68
	Merge sort	0,09	100,06
	Quick sort	0,01	100,06
	Shell sort	0,07	100,06

250.000	Bubble sort	378,04	99,71
	Selection sort	153,53	99,71
	Insertion sort	81,02	99,71
	Merge sort	0,19	100,70
	Quick sort	0,08	100,70
	Shell sort	0,16	100,70
500.000	Bubble sort	1520,88	18,70
	Selection sort	614,51	18,70
	Insertion sort	325,70	18,70
	Merge sort	0,38	19,05
	Quick sort	0,15	19,05
	Shell sort	0,34	19,05
1.000.000	Bubble sort	6041,59	32,55
	Selection sort	2487,51	11,63
	Insertion sort	1298,55	11,63
	Merge sort	0,77	12,72
	Quick sort	0,34	12,72
	Shell sort	0,71	12,72

Note: Proses compile pada data 1.500.000 dan 2.000.000 dihentikan karena telah memakan waktu hingga lebih dari 15 jam, namun hasil belum juga didapat

b. Data kata

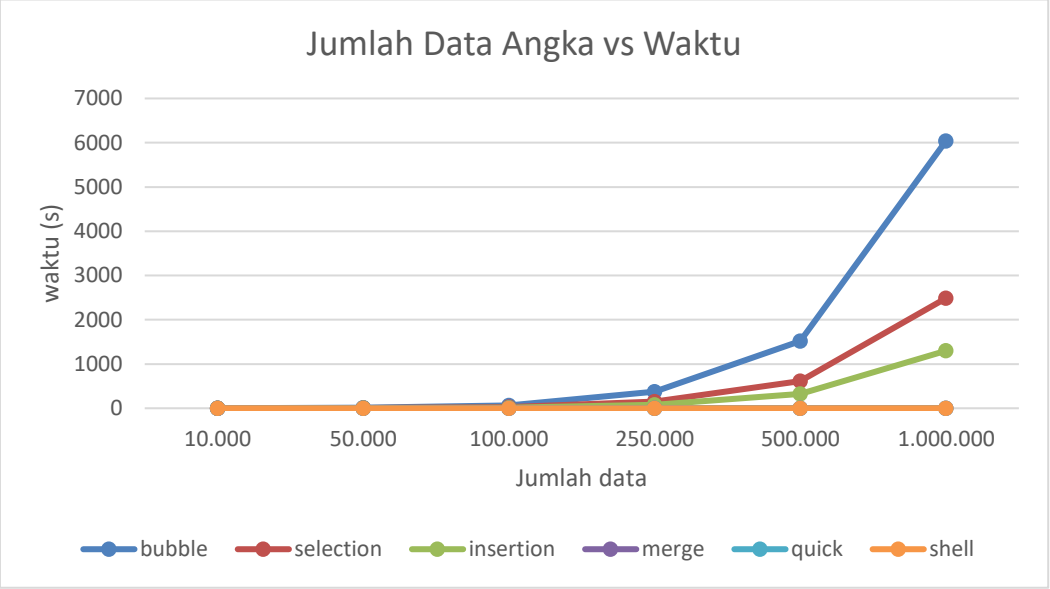
Jumlah data	Algoritma	Waktu	memory
10.000	Bubble sort	4,27	99,77
	Selection sort	0,46	99,77
	Insertion sort	1,28	99,77
	Merge sort	0,02	99,95
	Quick sort	0,03	99,95
	Shell sort	0,03	99,95
50.000	Bubble sort	114,40	99,86
	Selection sort	11,19	99,86
	Insertion sort	31,31	99,86

	Merge sort	0,14	99,68
	Quick sort	0,10	99,68
	Shell sort	10,16	99,68
100.000	Bubble sort	425,83	100,06
	Selection sort	43,83	100,06
	Insertion sort	124,96	100,06
	Merge sort	0,27	99,72
	Quick sort	0,23	99,72
	Shell sort	0,37	99,72
250.000	Bubble sort	2648,11	21,01
	Selection sort	280,97	21,01
	Insertion sort	778,25	16,14
	Merge sort	0,72	16,77
	Quick sort	0,61	16,77
	Shell sort	1,05	16,77
500.000	Bubble sort	15017,10	14,61
	Selection sort	1132,49	24,06
	Insertion sort	3093,78	24,06
	Merge sort	1,49	24,92
	Quick sort	1,30	24,92
	Shell sort	2,54	24,92
1.000.000	Bubble sort	43095,92	15,73
	Selection sort	4577,56	44,06
	Insertion sort	12658.05	24.02
	Merge sort	3.1	45.03
	Quick sort	2.76	45.03
	Shell sort	5.92	45.03

Note: Proses compile pada data 1.500.000 dan 2.000.000 dihentikan karena telah memakan waktu hingga lebih dari 15 jam, namun hasil belum juga didapat

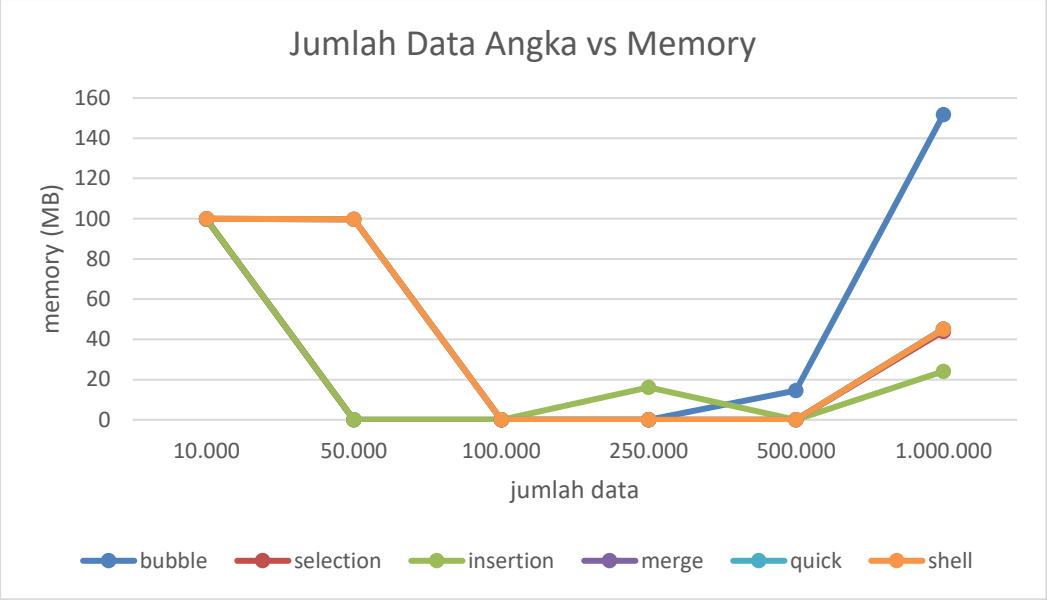
4. Grafik Perbandingan Waktu dan Memory

a. Data Angka



Gambar 7. Grafik perbandingan jumlah data angka dan waktu

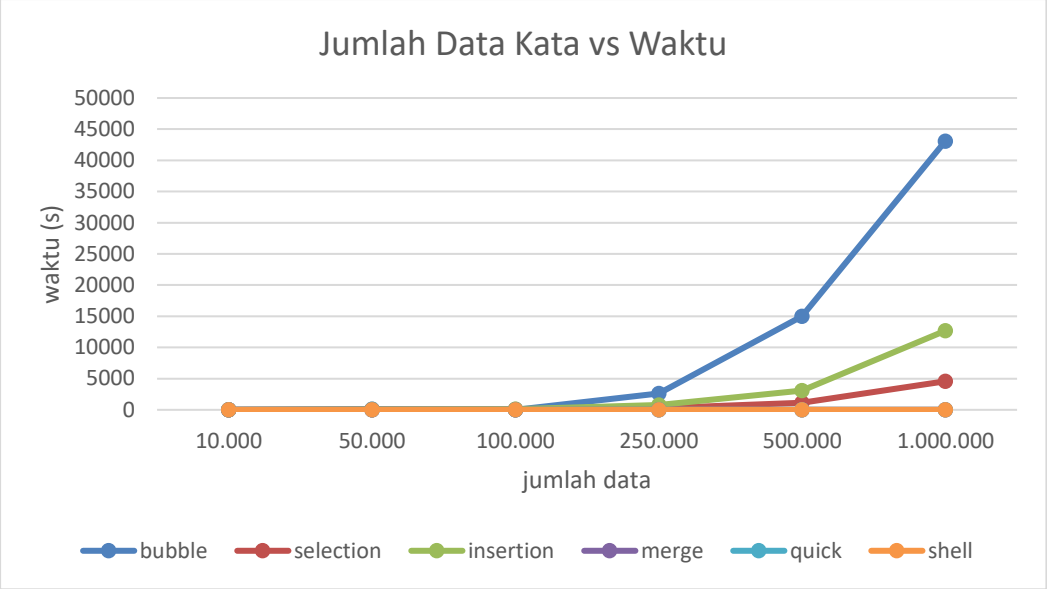
Grafik ini menunjukkan pertumbuhan waktu eksekusi (dalam detik) dari enam algoritma sorting seiring bertambahnya jumlah data (dari 10.000 sampai 1.000.000)**.** Sumbu X menunjukkan jumlah data, dan sumbu Y menunjukkan waktu (dalam detik)**.** Berdasarkan grafik, algoritma Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu yang drastis seiring bertambahnya jumlah data, mencapai ribuan detik pada 1.000.000 data. Sebaliknya, algoritma Quick Sort, Merge Sort, dan Shell Sort menunjukkan performa yang sangat efisien, dengan waktu eksekusi tetap rendah bahkan pada data besar. Perbedaan skala waktu ini menyebabkan garis algoritma cepat terlihat mendatar di grafik.



Gambar 8. Grafik perbandingan jumlah data angka dan memory

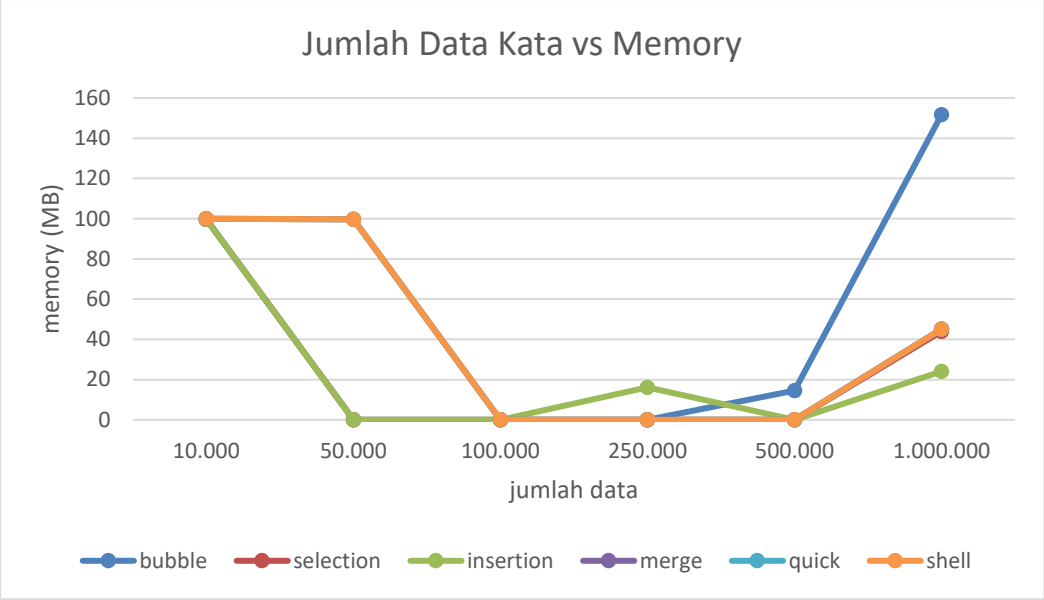
Grafik menunjukkan penggunaan memori (dalam MB**)** oleh masing-masing algoritma sorting saat memproses data dengan ukuran yang berbeda. Berdasarkan grafik perbandingan penggunaan memori, terlihat bahwa keenam algoritma sorting memiliki pola konsumsi memori yang relatif serupa pada ukuran data kecil hingga sedang (10.000 hingga 250.000). Namun, saat data bertambah, terlihat perbedaan yang signifikan. Bubble Sort menunjukkan lonjakan drastis di 1 juta data hingga 160 MB, menjadikannya algoritma paling boros memori. Selection dan Insertion Sort juga mengalami peningkatan memori, meskipun tidak sebesar Bubble. Sebaliknya, algoritma seperti Shell, Merge, dan Quick Sort cenderung lebih stabil dan efisien, dengan penggunaan memori tetap rendah bahkan pada skala besar.

Grafik yang terlihat hanya tiga garis terlihat dominan karena tiga lainnya memiliki pola hampir identik dan datar, membuatnya tertumpuk secara visual. Hal ini menunjukkan bahwa efisiensi memori tidak hanya bergantung pada kompleksitas waktu, tetapi juga strategi internal tiap algoritma.



Gambar 9. Grafik perbandingan jumlah data kata dan waktu

Berdasarkan grafik, algoritma Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu eksekusi yang sangat tajam seiring bertambahnya jumlah data, terutama pada ukuran 250.000 ke atas. Hal ini sejalan dengan teori kompleksitas waktu $O(n^2)$ yang membuat ketiganya tidak efisien untuk data dalam skala besar. Di sisi lain, Merge Sort dan Quick Sort tetap konsisten menunjukkan waktu eksekusi yang jauh lebih rendah, meskipun jumlah data bertambah secara signifikan. Hal ini mencerminkan keunggulan algoritma divide and conquer yang mereka gunakan, dengan kompleksitas waktu $O(n \log n)$. Shell Sort juga menunjukkan performa yang cukup stabil dan lebih baik dibandingkan algoritma $O(n^2)$, karena memanfaatkan pengurangan gap dalam perbandingan. Oleh karena itu, dari perspektif efisiensi waktu, algoritma dengan kompleksitas log-linear seperti Merge dan Quick Sort sangat disarankan untuk pengolahan data dalam jumlah besar.



Gambar 10. Grafik perbandingan jumlah data angka dan memory

Grafik ini memperlihatkan penggunaan memori enam algoritma sorting saat mengurutkan data kata dari 10.000 hingga 1.000.000 entri. Pada data kecil (≤100.000), semua algoritma menggunakan memori tinggi dan hampir sama (~99 MB). Bubble Sort menunjukkan peningkatan penggunaan memori yang sangat signifikan, menjadikannya algoritma paling boros dari segi memori. Di sisi lain, Insertion Sort dan Shell Sort menunjukkan efisiensi tinggi dengan memori yang stabil dan rendah, bahkan pada jumlah data besar.

Merge Sort dan Quick Sort, meskipun lebih efisien dibanding Bubble, tetap menggunakan memori yang lebih besar dibanding Shell dan Insertion karena struktur rekursif dan mekanisme alokasi tambahan yang digunakan. Hanya tiga garis terlihat jelas dalam grafik karena tiga algoritma lainnya memiliki penggunaan memori sangat mirip dan rendah, sehingga garis-garisnya saling bertumpuk dan tampak mendatar. Perbedaan skala inilah yang membuat algoritma efisien seperti Shell dan Insertion tampak tersembunyi, meskipun performanya justru lebih baik untuk data besar.

5. Analisis dan kesimpulan

a. Analisis

Perbandingan antara data angka dan data kata menunjukkan bahwa jenis data memiliki pengaruh yang signifikan terhadap performa waktu dan memori dari

masing-masing algoritma sorting. Secara umum, baik pada data angka maupun kata, algoritma dengan kompleksitas waktu $O(n^2)$ seperti Bubble Sort**,** Selection Sort**,** dan Insertion Sort menunjukkan peningkatan waktu eksekusi yang sangat tajam seiring bertambahnya jumlah data. Namun, waktu yang dibutuhkan untuk mengurutkan data kata cenderung sedikit lebih tinggi dibandingkan data angka karena operasi perbandingan string secara leksikografis lebih kompleks daripada angka. Di sisi lain, algoritma efisien seperti Quick Sort dan Merge Sort tetap menunjukkan performa waktu yang stabil dan rendah pada kedua jenis data, karena mereka mengimplementasikan strategi divide and conquer yang efisien untuk berbagai tipe data.

Dari sisi penggunaan memori, baik pada data angka maupun kata, penggunaan memori secara umum stabil pada jumlah data kecil, namun mulai menunjukkan variasi signifikan pada data besar. Bubble Sort cenderung menjadi algoritma yang paling boros memori, terutama pada data kata, di mana penggunaannya mencapai lebih dari 150 MB. Sebaliknya, Shell Sort dan Insertion Sort justru tampil unggul dalam efisiensi memori pada data besar, terutama untuk data kata, dengan penggunaan memori yang tetap rendah dan stabil. Hal ini menunjukkan bahwa selain kompleksitas waktu, struktur data yang diurutkan dan strategi internal algoritma juga sangat memengaruhi efisiensi eksekusi, baik dari segi kecepatan maupun konsumsi sumber daya memori. Dengan demikian, pemilihan algoritma sorting yang tepat perlu mempertimbangkan jenis data dan skala pengolahan untuk mendapatkan hasil yang optimal.

b. Kesimpulan

Berdasarkan hasil pengujian enam algoritma sorting terhadap data angka dan data kata dengan berbagai ukuran, dapat disimpulkan bahwa efisiensi algoritma sangat dipengaruhi oleh kompleksitas waktu dan strategi internal masing-masing algoritma. Algoritma sederhana seperti Bubble Sort**,** Selection Sort**,** dan Insertion Sort menunjukkan performa yang buruk pada skala data besar, dengan waktu eksekusi yang meningkat secara drastis karena kompleksitas waktu $O(n^2)$**.** Sebaliknya, Merge Sort dan Quick Sort terbukti jauh lebih efisien dalam waktu eksekusi karena mengimplementasikan pendekatan divide and conquer dengan kompleksitas $O(n \log n)$**,** membuatnya ideal untuk pemrosesan data dalam jumlah besar.

Dalam hal penggunaan memori, sebagian besar algoritma menunjukkan

konsumsi memori yang stabil pada data kecil hingga sedang, namun pada data besar terdapat variasi yang mencolok. Bubble Sort secara konsisten menjadi algoritma yang paling boros memori, terutama saat mengurutkan data kata, sementara Insertion Sort dan Shell Sort terlihat lebih efisien dan stabil dalam penggunaan memori.

Selain itu, perbedaan tipe data (angka dan kata) juga memberikan dampak terhadap performa, khususnya pada waktu eksekusi, di mana pengurutan string cenderung memerlukan waktu lebih lama dibanding angka karena perbandingan leksikografis yang lebih kompleks. Dengan demikian, pemilihan algoritma sorting harus disesuaikan dengan jenis data, ukuran data, serta prioritas antara efisiensi waktu dan penggunaan memori**,** untuk mendapatkan hasil yang optimal dalam implementasi nyata.