Laporan Mata Kuliah Struktur Data dan Algoritma

SIMPLE SORTING DAN ADVANCE SORTING

disusun untuk memenuhi

tugas mata kuliah Struktur Data dan Algoritma

Oleh:

MUHAMMAD FARUQI

(2308107010005)

JURUSAN INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS SYIAH KUALA

DARUSSALAM, BANDA ACEH

2025

BAB I PENDAHULUAN

1.1 Latar Belakang

Algoritma pengurutan (sorting) merupakan salah satu fondasi terpenting dalam ilmu komputer dan pengembangan perangkat lunak. Dalam era digital yang ditandai dengan pertumbuhan data yang eksponensial, kemampuan untuk mengurutkan informasi secara efisien menjadi semakin krusial. Pengurutan data tidak hanya sekadar mengatur elemen-elemen dalam urutan tertentu, tetapi juga menjadi prasyarat untuk berbagai operasi pemrosesan data lainnya seperti pencarian, penggabungan, dan analisis.

Pentingnya algoritma sorting dalam dunia pemrograman dapat dilihat dari penggunaannya yang sangat luas di berbagai domain aplikasi. Dari sistem basis data yang menangani jutaan transaksi per detik hingga aplikasi mobile yang harus merespons dengan cepat meski dengan sumber daya terbatas, algoritma sorting menjadi komponen kritis yang menentukan responsivitas dan keandalan sistem. Selain itu, pada aplikasi dengan dataset besar seperti big data analytics, kecepatan pemrosesan data sangat bergantung pada efisiensi algoritma sorting yang diimplementasikan.

Efisiensi algoritma sorting memiliki dampak langsung terhadap performa aplikasi secara keseluruhan. Algoritma yang tidak efisien dapat menyebabkan penggunaan sumber daya komputasi yang berlebihan, seperti CPU dan memori, yang mengakibatkan aplikasi berjalan lambat atau bahkan tidak responsif. Sebagai contoh, perbedaan antara algoritma dengan kompleksitas $O(n^2)$ seperti Bubble Sort dan algoritma dengan kompleksitas $O(n \log n)$ seperti Quick Sort atau Merge Sort dapat sangat signifikan ketika bekerja dengan dataset besar. Pada kasus tertentu, pemilihan algoritma sorting yang tepat dapat mempercepat proses hingga ratusan kali lipat.

Di era komputasi modern saat ini, dengan meningkatnya jumlah perangkat yang terhubung dan aplikasi yang berjalan pada berbagai platform, optimasi kinerja menjadi hal yang tidak bisa ditawar. Setiap mikrodetik penghematan dalam pemrosesan data dapat berdampak signifikan pada pengalaman pengguna, konsumsi daya, dan biaya operasional pada skala besar. Pengembang yang menguasai pemilihan dan implementasi algoritma sorting yang sesuai dengan konteks aplikasi spesifik akan mampu menghasilkan solusi perangkat lunak yang tidak hanya fungsional tetapi juga optimal dari segi kinerja dan efisiensi sumber daya.

1.2 Tujuan

1. Mengimplementasikan enam algoritma sorting (Bubble Sort, Selection, Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort) untuk mengurutkan data bertipe angka dan kata menggunakan Bahasa C.
2. Mengukur dan membandingkan efisiensi waktu eksekusi dan penggunaan memori dari setiap algoritma pada berbagai ukuran data, mulai dari 10.000 hingga 2.000.000 baris.
3. Menganalisis performa relatif tiap algoritma berdasarkan hasil eksperimen untuk menentukan algoritma paling efisien sesuai jenis dan jumlah data yang diolah.

BAB II DESKRIPSI ALGORITMA

2.1 Bubble Sort

Bubble Sort merupakan algoritma pengurutan sederhana yang bekerja dengan cara membandingkan dan menukar pasangan elemen bersebelahan secara berulang hingga seluruh array terurut. Pada setiap iterasi, elemen terbesar akan "menggelembung" ke posisi akhir. Meskipun mudah diimplementasikan, Bubble Sort memiliki kompleksitas waktu $O(n^2)$ pada kasus terburuk dan rata-rata, yang membuatnya tidak efisien untuk dataset besar, namun masih bisa optimal pada data yang hampir terurut dengan pengecekan early termination.

2.2 Selection Sort

Selection Sort menyortir elemen dengan cara memilih elemen terkecil (atau terbesar) dari sisa array yang belum disortir, lalu menukarnya dengan elemen di posisi saat ini. Proses ini berulang untuk setiap posisi dalam array. Selection Sort memiliki kompleksitas waktu $O(n^2)$, sama seperti Bubble Sort, namun jumlah pertukarannya lebih sedikit, sehingga kadang sedikit lebih efisien dalam implementasi tertentu.

2.3 Insertion Sort

Insertion Sort menyusun data dengan mengambil satu elemen dari array dan menyisipkannya ke posisi yang tepat dalam bagian array yang sudah tersortir. Metode ini sangat efisien untuk dataset kecil atau hampir terurut, dengan kompleksitas waktu terbaik $O(n)$ dan terburuk $O(n^2)$. Karena pendekatannya yang adaptif, Insertion Sort sering digunakan dalam kasus real-time sorting dengan data kecil.

2.4 Merge Sort

Merge Sort adalah algoritma berbasis pendekatan divide-and-conquer yang membagi array menjadi dua bagian, mengurutkan kedua bagian secara rekursif, kemudian menggabungkan (merge) kedua bagian terurut tersebut. Dengan kompleksitas waktu $O(n \log n)$ pada semua kasus, Merge Sort menawarkan performa yang stabil dan dapat diandalkan terlepas dari distribusi data, meskipun membutuhkan ruang memori tambahan $O(n)$ untuk proses penggabungan, membuatnya menjadi pilihan yang baik untuk pengurutan data dalam jumlah besar.

2.5 Quick Sort

Quick Sort juga merupakan algoritma divide-and-conquer yang bekerja dengan memilih elemen pivot, membagi array menjadi dua sub-array (elemen yang lebih kecil dan lebih besar dari pivot), kemudian secara rekursif mengurutkan sub-array tersebut. Quick Sort memiliki kompleksitas waktu rata-rata $O(n \log n)$ dan merupakan salah satu algoritma pengurutan paling efisien dalam praktik karena konstanta waktu yang rendah dan penggunaan memori yang efisien, meskipun memiliki kasus terburuk $O(n^2)$ ketika pivot yang dipilih tidak optimal.

2.6 Shell Sort

Shell Sort merupakan penyempurnaan dari Insertion Sort yang menyortir elemen berdasarkan jarak atau gap tertentu, kemudian mengecilkan gap secara bertahap hingga menjadi 1. Teknik ini mempercepat proses penyisipan karena elemen bisa berpindah jarak jauh lebih awal. Walaupun kompleksitas waktunya bervariasi tergantung strategi gap, Shell Sort lebih efisien daripada Bubble, Selection, atau Insertion Sort untuk dataset sedang.

BAB III STRATEGI IMPLEMENTASI

3.1 Bahasa dan Perangkat Lunak

Program ini diimplementasikan menggunakan bahasa pemrograman **C** karena kemampuannya dalam mengelola memori secara langsung serta efisiensinya dalam pemrosesan data. Proses kompilasi dilakukan menggunakan **GCC (GNU Compiler Collection)** pada sistem operasi **Windows**, dengan bantuan library tambahan psapi.h untuk membaca informasi penggunaan memori dari proses yang berjalan. Proses pengembangan dilakukan menggunakan editor **Visual Studio Code**.

3.2 Struktur File Program

Program ini disusun secara modular dalam tiga file utama:

- **`main.c`** – berisi fungsi utama (**`main()`**) yang menangani alur antarmuka pengguna dan pemilihan menu.
- **`functions.c`** – berisi implementasi semua fungsi sorting, pembacaan file, pengukuran waktu, dan memori.
- **`header.h`** – berisi deklarasi fungsi-fungsi yang digunakan di main.c dan functions.c.

3.3 Alur Program

1. Pengguna memilih jenis data yang akan disorting, yaitu **angka** atau **kata**.
2. Jumlah baris data yang akan digunakan ditentukan oleh pengguna. Untuk angka, data akan di-*generate* sebagai angka acak, dan untuk kata, akan di-*generate* sebagai string acak dengan panjang tertentu.
3. Program menyimpan data tersebut ke dalam file teks (data_angka.txt atau data_kata.txt) dan membaca kembali isinya ke dalam array untuk proses sorting.
4. Pengguna kemudian memilih salah satu dari enam metode sorting: Bubble, Selection, Insertion, Merge, Quick, atau Shell Sort.
5. Program melakukan proses sorting sesuai metode yang dipilih, lalu mencatat **waktu eksekusi** dan **penggunaan memori** selama proses berlangsung.
6. Hasil sorting ditampilkan (10 data pertama), bersama dengan waktu dan memori yang digunakan.
7. Data waktu dan memori direkam dan dicatat secara manual oleh pengguna untuk keperluan analisis dan visualisasi grafik.

3.4 Pengukuran Waktu dan Memori

- **Waktu eksekusi** dihitung menggunakan fungsi **`clock()`** dari **`time.h`**, dengan perhitungan selisih waktu sebelum dan sesudah proses sorting.
- **Penggunaan memori** diukur menggunakan fungsi **`GetProcessMemoryInfo()`** dari `psapi.h`, yang mengambil data **`WorkingSetSize`** sebagai estimasi jumlah memori fisik yang digunakan selama proses sorting berlangsung.

3.5 Penanganan Memori

Semua alokasi memori dilakukan secara dinamis menggunakan **`malloc()`** agar dapat menangani data dalam jumlah besar (hingga 2 juta baris). Program juga melakukan **`free()`** pada semua pointer yang dialokasikan untuk mencegah kebocoran memori (*memory leak*).

BAB IV HASIL EKSPERIMEN DAN VISUALISASI DATA

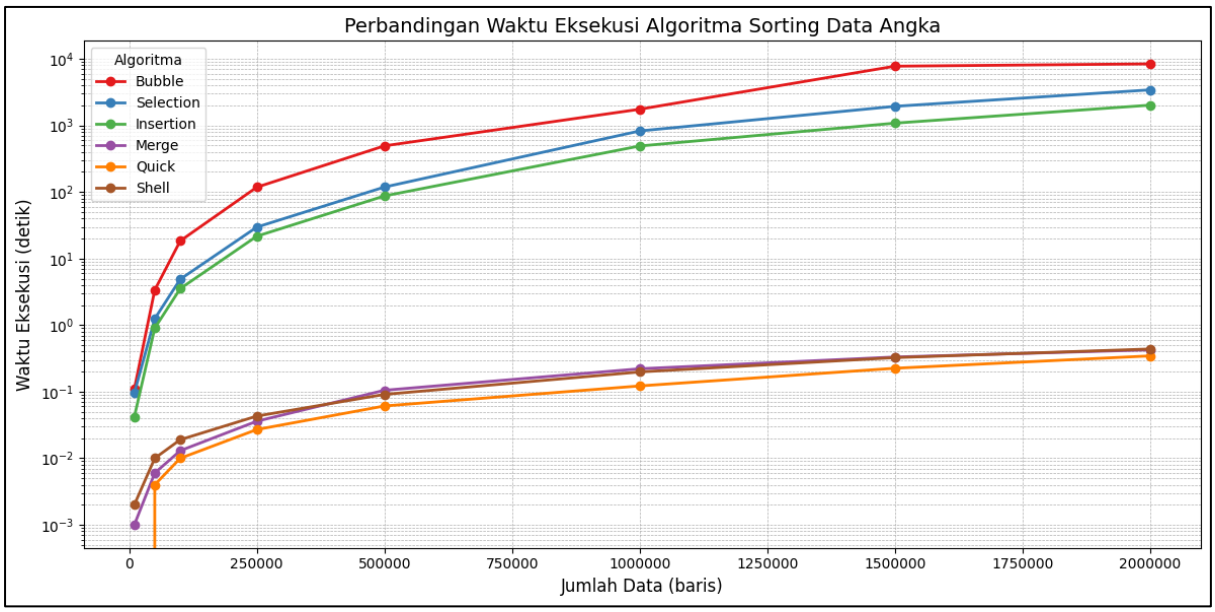
4.1 Hasil Sorting Data Angka

Setelah dilakukan pengujian terhadap enam algoritma pengurutan pada data angka dengan jumlah baris bervariasi, diperoleh hasil waktu eksekusi dan penggunaan memori yang menggambarkan perbedaan efisiensi masing-masing metode. Hasil tersebut disajikan dalam bentuk tabel dan grafik untuk mempermudah analisis visual.

a. Waktu Ekseskusi

Waktu Eksekusi (detik)								
Algoritma	Ukuran data (baris)							
	10000	50000	100000	250000	500000	1000000	1500000	2000000
Bubble	0.11	3.374	18.446	118.209	493.795	1751.578	7769.198	8412.293
Selection	0.095	1.248	4.9	29.701	118.728	826.13	1937.811	3424.873
Insertion	0.042	0.902	3.54	21.677	87.305	491.209	1082.62	2017.876
Merge	0.001	0.006	0.013	0.036	0.105	0.221	0.332	0.425
Quick	0	0.004	0.01	0.027	0.061	0.122	0.225	0.345
Shell	0.002	0.01	0.019	0.043	0.091	0.198	0.325	0.438

Tabel 4.1 Tabel *Pencatatan Waktu Eksekusi Algoritma Sorting Data Angka*



Gambar 4.1 *Grafik Perbandingan Waktu Eksekusi Algoritma Sorting Data Angka*

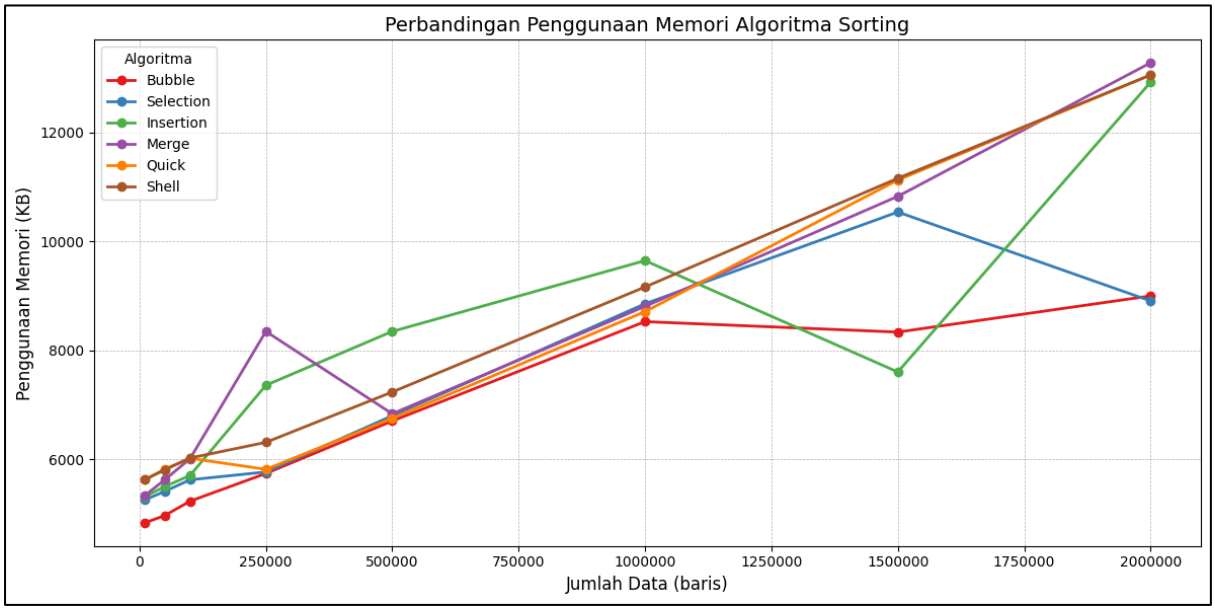
Tabel 4.1 menampilkan waktu eksekusi enam algoritma dalam menyortir data angka pada berbagai ukuran. Dari data tersebut, terlihat bahwa algoritma dengan kompleksitas waktu $O(n^2)$, seperti Bubble Sort, Selection Sort, dan Insertion Sort, menunjukkan pertumbuhan waktu yang sangat cepat seiring bertambahnya ukuran data. Bahkan, ketiganya gagal menyelesaikan sorting pada ukuran data 1.000.000 baris ke atas.

Hal ini semakin dipertegas oleh Gambar 4.1 yang menunjukkan kurva waktu eksekusi setiap algoritma. Quick Sort, Merge Sort, dan Shell Sort menghasilkan grafik yang cenderung landai, menandakan efisiensi tinggi terhadap pertambahan data. Quick Sort konsisten menjadi metode tercepat, sedangkan Merge Sort dan Shell Sort juga menunjukkan stabilitas waktu yang baik.

b. Penggunaan Memori

Penggunaan Memori (KB)								
Algoritma	**Ukuran data (baris)**							
	10000	**50000**	**100000**	**250000**	**500000**	**1000000**	**1500000**	**2000000**
Bubble	4832	4968	5232	5740	6704	8528	8336	9000
Selection	5252	5412	5624	5768	6800	8856	10540	8908
Insertion	5312	5500	5704	7360	8348	9648	7604	12924
Merge	5332	5624	6012	8348	6836	8808	10828	13280
Quick	5624	5820	6020	5816	6752	8712	11128	13056
Shell	5624	5808	6024	6312	7236	9164	11160	13056

Tabel 4.2 *Tabel Pencatatan Penggunaan Memori Algoritma Sorting Data Angka*



Gambar 4.2 *Grafik Perbandingan Penggunaan Memori Algoritma Sorting Data Angka*

Tabel 4.2 memperlihatkan penggunaan memori oleh masing-masing algoritma saat menyortir data angka. Merge Sort tercatat sebagai algoritma dengan konsumsi memori terbesar karena memerlukan array tambahan untuk proses penggabungan. Di sisi lain, Quick Sort dan Shell Sort memanfaatkan memori secara lebih efisien.

Gambar 4.2 menunjukkan tren linear pada penggunaan memori untuk semua algoritma, namun tetap terlihat perbedaan antar metode. Meskipun algoritma seperti Bubble, Selection, dan Insertion Sort menunjukkan penggunaan memori yang lebih kecil, keunggulan ini tidak cukup untuk menutupi kelemahan utamanya, yaitu waktu eksekusi yang terlalu lama.

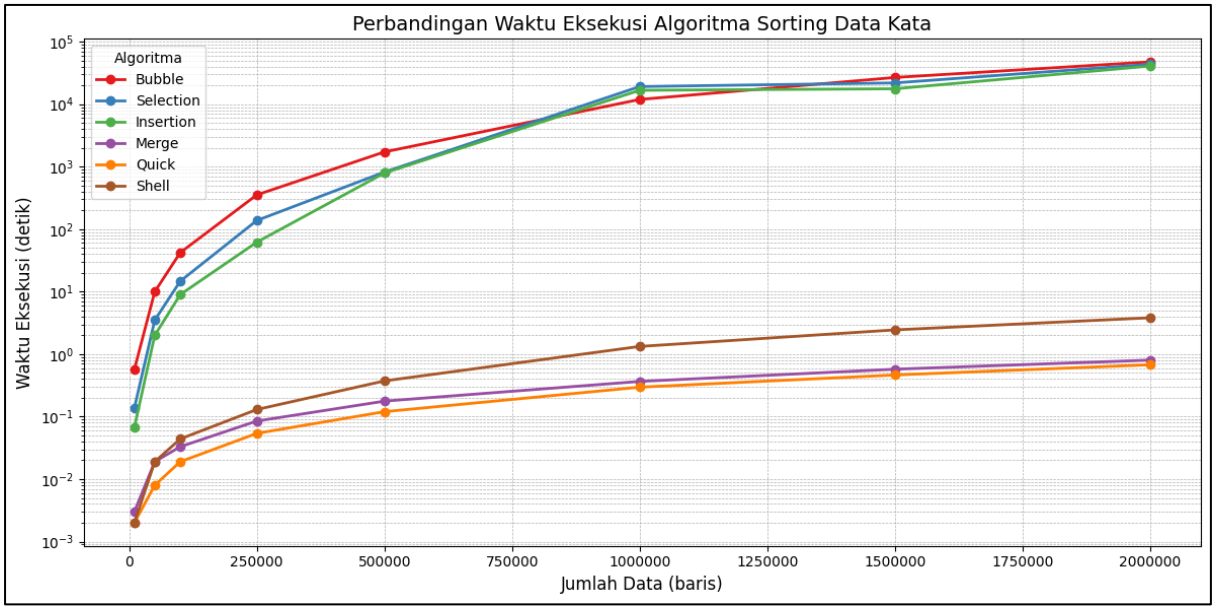
4.2 Hasil Sorting Data Kata

Setelah dilakukan pengujian pada data berupa kata acak dengan panjang karakter yang bervariasi, diperoleh hasil performa algoritma dalam hal waktu eksekusi dan penggunaan memori. Sama seperti pengujian data angka, hasil disajikan dalam bentuk tabel dan grafik sebagai dasar untuk membandingkan efisiensi setiap algoritma.

a. Waktu Eksekusi

Waktu Running (Detik)								
Algoritma	**Ukuran data (baris)**							
	10000	**50000**	**100000**	**250000**	**500000**	**1000000**	**1500000**	**2000000**
Bubble	0.568	9.986	42.16	354.095	1722.06	11898.002	26758.4	47512.35
Selection	0.137	3.499	14.724	137.853	819.979	19138.163	21893.54	43514.98
Insertion	0.067	1.999	8.98	61.94	792.486	16687.506	17642.554	40789.213
Merge	0.003	0.019	0.033	0.085	0.177	0.366	0.574	0.802
Quick	0.002	0.008	0.019	0.054	0.12	0.296	0.464	0.676
Shell	0.002	0.019	0.044	0.13	0.373	1.328	2.444	3.822

Tabel 4.3 *Tabel Pencatatan Waktu Eksekusi Algoritma Sorting Data Kata*



Gambar 4.3 *Grafik Perbandingan Waktu Eksekusi Algoritma Sorting Data Kata*

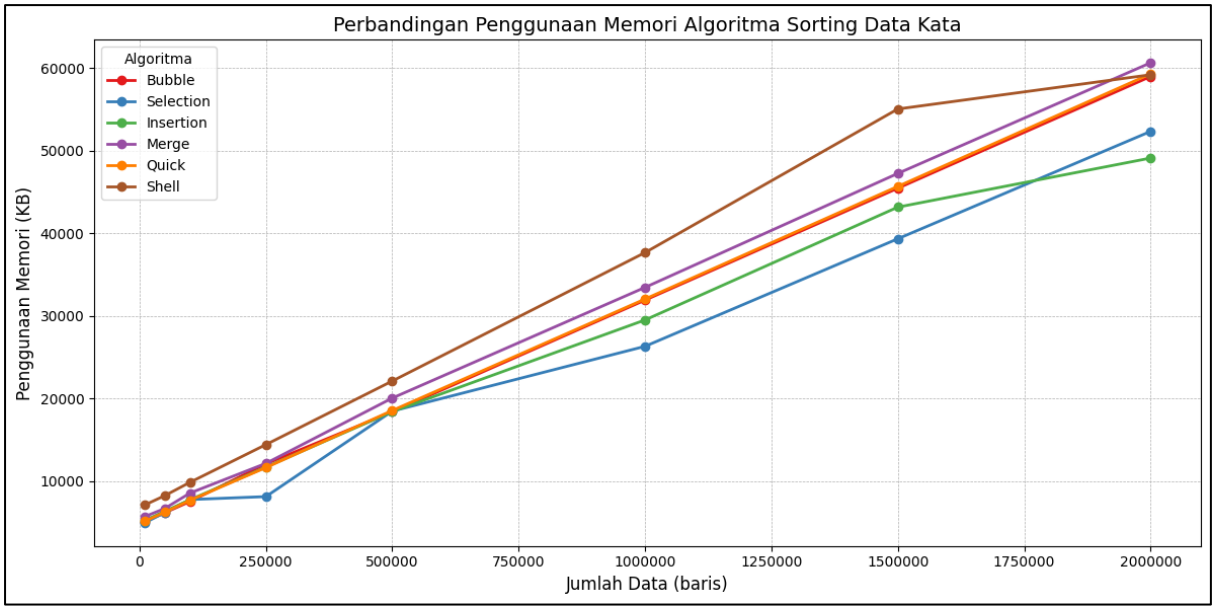
Tabel 4.3 menyajikan waktu eksekusi enam algoritma dalam menyortir data berupa kata acak dengan panjang antara 3 hingga 20 karakter. Hasil yang diperoleh menunjukkan pola performa yang serupa dengan pengurutan data angka. Quick Sort kembali menjadi algoritma dengan waktu eksekusi tercepat dan paling konsisten pada semua ukuran data. Merge Sort dan Shell Sort juga menunjukkan performa efisien, dengan grafik yang cenderung landai hingga pada jumlah data 2.000.000 kata.

Sebaliknya, algoritma dengan kompleksitas O(n^2) seperti Bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu yang sangat signifikan seiring bertambahnya jumlah data. Meskipun tidak ada proses yang gagal atau terhenti, waktu eksekusi untuk algoritma ini jauh melampaui metode lainnya pada ukuran data besar, sebagaimana terlihat pada Gambar 4.3. Hal ini menegaskan bahwa efisiensi algoritma sangat memengaruhi skalabilitas dalam konteks data non-numerik seperti string.

b. Penggunaan Memori

Penggunaan Memori (KB)								
Algoritma	**Ukuran data (baris)**							
	10000	**50000**	**100000**	**250000**	**500000**	**1000000**	**1500000**	**2000000**
Bubble	5092	6168	7536	12092	18384	31921	45500	59000
Selection	4936	6188	7768	8124	18476	26324	39347	52351
Insertion	5220	6352	7748	11696	18416	29516	43180	49123
Merge	5692	6688	8556	12148	20056	33460	47260	60660
Quick	5124	6236	7680	11636	18552	32052	45692	59320
Shell	7100	8240	9860	14420	22120	37676	55064	59212

Tabel 4.4 *Tabel Pencatatan Penggunaan Memori Algoritma Sorting Data Kata*



Gambar 4.4 *Grafik Perbandingan Penggunaan Memori Algoritma Sorting Data Kata*

Penggunaan memori masing-masing algoritma selama proses pengurutan data kata ditampilkan dalam Tabel 4.4. Secara umum, jumlah memori yang digunakan lebih tinggi dibanding saat menyortir data angka. Hal ini disebabkan oleh struktur data string yang memerlukan alokasi ruang memori lebih besar, baik untuk menyimpan pointer ke string maupun isi karakter dari kata itu sendiri.

Seperti pada data angka, Merge Sort menunjukkan konsumsi memori tertinggi karena memanfaatkan array tambahan selama proses penggabungan. Algoritma lain seperti Bubble, Selection, dan Insertion Sort tampak menggunakan memori lebih rendah, tetapi memiliki waktu eksekusi yang jauh lebih tinggi, sehingga tetap tidak efisien secara keseluruhan. Gambar 4.4 menunjukkan tren pertumbuhan penggunaan memori yang linear terhadap ukuran data, dengan perbedaan antar algoritma yang konsisten di seluruh skenario.

BAB V ANALISIS PERBANDINGAN EFISIENSI ALGORITMA

5.1 Analisis Waktu Ekseskusi

Grafik pada Bab IV menunjukkan perbedaan signifikan antara algoritma $O(n^2)$ (Bubble, Selection, Insertion) dan O(n log n) (Merge, Quick, Shell). Untuk data hingga 100.000 baris, seluruh algoritma masih cepat, namun mulai dari 250.000 baris ke atas, algoritma $O(n^2)$ mengalami lonjakan waktu yang drastis.

Pada 500.000 baris, Bubble Sort memerlukan lebih dari 8 menit, sedangkan Quick dan Merge Sort hanya sekitar 0.2 detik. Di atas 1 juta baris, algoritma $O(n^2)$ tidak dapat menyelesaikan proses dalam waktu wajar. Quick Sort menjadi yang tercepat, diikuti Merge dan Shell Sort yang juga stabil.

5.2 Analisis Penggunaan Memori

Dalam aspek penggunaan memori, algoritma seperti **Merge Sort** menggunakan lebih banyak memori dibanding algoritma lainnya, karena membutuhkan array tambahan dalam proses penggabungan (merge). Hal ini terlihat dari lonjakan memori Merge Sort yang mencapai lebih dari 13 MB pada ukuran 2 juta baris. Quick Sort dan Shell Sort memiliki penggunaan memori yang lebih efisien karena tidak memerlukan struktur data tambahan yang besar, meskipun Quick Sort masih menggunakan stack untuk rekursi.

Sementara itu, algoritma seperti Bubble, Selection, dan Insertion terlihat memiliki penggunaan memori yang lebih rendah. Namun**,** memori yang rendah tidak selalu berarti efisiensi yang lebih baik, terutama jika harus dikompensasi dengan waktu eksekusi yang sangat tinggi. Dalam konteks efisiensi keseluruhan, algoritma dengan waktu eksekusi yang cepat dan memori yang masih dalam batas wajar (seperti Quick dan Merge Sort) tetap menjadi pilihan utama.

5.3 Efisiensi pada Data Kata

Pengujian juga dilakukan pada data berupa kata acak. Hasilnya menunjukkan bahwa proses sorting kata umumnya membutuhkan waktu yang lebih lama dibandingkan data angka, terutama karena proses perbandingan string (**strcmp**) yang lebih kompleks. Pola efisiensi masih serupa dengan data angka: Quick Sort dan Merge Sort tetap menjadi yang tercepat, sedangkan Bubble, Selection, dan Insertion mengalami keterlambatan signifikan, dan sampai memakan waktu 3 jam lebih untuk mengurutkan 1.000.000 kata ke atas.

5.4 Rekomendasi dan Penggunaan

Berdasarkan hasil dan analisis di atas, rekomendasi pemilihan algoritma dapat disimpulkan sebagai berikut:

- Untuk **data berukuran kecil (≤ 50.000)**: seluruh algoritma dapat digunakan, namun Insertion Sort lebih unggul jika data hampir terurut.
- Untuk **data sedang (50.000–250.000)**: disarankan menggunakan Quick Sort atau Shell Sort.
- Untuk **data besar (≥ 500.000)**: **Quick Sort** dan **Merge Sort** sangat direkomendasikan karena efisien dalam waktu dan stabil dalam performa.
- Penggunaan **Bubble, Selection, dan Insertion Sort** tidak disarankan untuk data besar karena tidak efisien secara waktu, meskipun penggunaan memorinya lebih rendah.

BAB VI KESIMPULAN

6.1 Kesimpulan

Berdasarkan hasil eksperimen, dapat disimpulkan bahwa algoritma Bubble Sort, Selection Sort, dan Insertion Sort tidak efisien untuk data berukuran besar karena waktu eksekusinya meningkat drastis, bahkan gagal menyelesaikan sorting pada data ≥ 1.000.000 baris. Meskipun penggunaan memorinya lebih rendah, performanya tidak layak untuk skala besar.

Sebaliknya, algoritma Quick Sort, Merge Sort, dan Shell Sort menunjukkan kinerja yang konsisten dan efisien, baik dari segi waktu maupun memori. Quick Sort menjadi yang tercepat, sementara Merge Sort unggul dalam stabilitas, meskipun menggunakan memori lebih besar.

Pola efisiensi serupa juga terjadi pada data berupa kata, dengan algoritma $O(n \log n)$ tetap unggul. Oleh karena itu, untuk dataset berukuran besar, hanya algoritma dengan efisiensi tinggi yang dapat diandalkan dalam praktik.

6.2 Saran

Berdasarkan hasil eksperimen dan analisis yang telah dilakukan, disarankan untuk menggunakan algoritma **Quick Sort** atau **Merge Sort** dalam pengurutan data berukuran besar karena terbukti memiliki efisiensi tinggi dan waktu eksekusi yang stabil. Untuk data berukuran kecil hingga sedang, **Insertion Sort** masih dapat menjadi pilihan, terutama jika data sudah hampir terurut. Sementara itu, algoritma seperti **Bubble Sort** dan **Selection Sort** sebaiknya hanya digunakan untuk keperluan edukasi atau demonstrasi dasar sorting, mengingat performanya yang sangat terbatas pada skala data besar. Ke depan, eksperimen dapat dikembangkan lebih lanjut dengan jenis data yang lebih kompleks atau realistis, seperti data semi-terurut atau data dunia nyata, guna menguji kestabilan dan efisiensi algoritma dalam kondisi yang lebih bervariasi.