Laporan Mata Kuliah Struktur Data dan Algoritma

TUGAS 4 Sorting Algorithms

Disusun untuk memenuhi tugas Mata Kuliah
Struktur Data dan Algoritma

Oleh:

Muhammad Azani Irvand
(2308107010026)

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

1. Pendahuluan

Tugas ini bertujuan untuk melakukan implementasi dan analisis performa dari beberapa algoritma pengurutan (sorting) fundamental dalam Struktur Data dan Algoritma. Algoritma yang dianalisis meliputi Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort, dan Shell Sort. Analisis difokuskan pada dua metrik utama: waktu eksekusi dan penggunaan memori. Tujuan spesifik dari tugas ini adalah

- Mengevaluasi performa (waktu eksekusi dan penggunaan memori) berbagai algoritma sorting dalam menangani dataset berskala besar, baik berupa data integer maupun string.
- Memahami kompleksitas waktu dari masing-masing algoritma secara teori dan membandingkannya dengan hasil eksperimen nyata.

Eksperimen dilakukan pada data acak dengan jumlah hingga 2.000.000 item untuk melihat bagaimana skalabilitas masing-masing algoritma.

2. Deskripsi Algoritma

Berikut adalah deskripsi singkat mengenai prinsip kerja dan kompleksitas waktu teoretis dari masing-masing algoritma sorting yang diimplementasikan:

2.1 Bubble Sort

Prinsip Kerja: Membandingkan pasangan elemen yang berdekatan secara berulang dan menukarnya jika urutannya salah. Proses ini diulangi hingga tidak ada lagi pertukaran dalam satu iterasi penuh.

Kompleksitas Waktu (Big O):

- Kasus Terbaik: O(n) - jika data sudah terurut.
- Kasus Rata-rata: O(n^2)
- Kasus Terburuk: O(n^2)
- Kompleksitas Ruang: O(1) (in-place).

2.2 Selection Sort

Prinsip Kerja: Membagi array menjadi dua bagian: terurut dan tidak terurut.

Secara berulang mencari elemen minimum (atau maksimum) dari bagian tidak terurut dan memindahkannya ke akhir bagian terurut.

Kompleksitas Waktu (Big O):

- Kasus Terbaik: O(n^2)
- Kasus Rata-rata: O(n^2)
- Kasus Terburuk: O(n^2)
- Kompleksitas Ruang: O(1) (in-place).

2.3 Insertion Sort

Prinsip Kerja: Membangun array terurut satu elemen pada satu waktu. Mengambil elemen dari bagian tidak terurut dan menyisipkannya ke posisi yang tepat di bagian yang sudah terurut.

Kompleksitas Waktu (Big O):

- Kasus Terbaik: O(n) - jika data sudah terurut.
- Kasus Rata-rata: O(n^2)
- Kasus Terburuk: O(n^2)
- Kompleksitas Ruang: O(1) (in-place).

2.4 Merge Sort

Prinsip Kerja: Algoritma divide-and-conquer. Membagi array menjadi dua bagian secara rekursif hingga tersisa satu elemen (dianggap terurut), kemudian menggabungkan (merge) kembali sub-array yang sudah terurut.

Kompleksitas Waktu (Big O)

- Kasus Terbaik: O(n log n)
- Kasus Rata-rata: O(n log n)
- Kasus Terburuk: O(n log n)
- Kompleksitas Ruang: O(n) - karena memerlukan array tambahan untuk proses penggabungan.

2.5 Quick Sort

Prinsip Kerja: Algoritma divide-and-conquer. Memilih sebuah elemen sebagai

'pivot', kemudian mempartisi array sehingga elemen yang lebih kecil dari pivot berada di kiri dan yang lebih besar di kanan. Proses ini diulangi secara rekursif pada sub-array kiri dan kanan.

Kompleksitas Waktu (Big O):

- Kasus Terbaik: O(n log n)
- Kasus Rata-rata: O(n log n)
- Kasus Terburuk: O(n^2) - terjadi jika pivot selalu elemen terkecil/terbesar.
- Kompleksitas Ruang: O(log n) (rata-rata, untuk stack rekursi) hingga O(n) (kasus terburuk). Implementasi in-place.

2.6 Shell Sort

Prinsip Kerja: Perbaikan dari Insertion Sort. Membandingkan dan menukar elemen yang berjauhan dengan jarak (gap) tertentu, kemudian secara bertahap mengurangi gap hingga menjadi 1 (menjadi Insertion Sort biasa). Menggunakan urutan gap (misalnya, Knuth sequence).

Kompleksitas Waktu (Big O): Sangat bergantung pada urutan gap yang digunakan.

- Kasus Terbaik: O(n log n) (tergantung gap)
- Kasus Rata-rata: Bervariasi, bisa mendekati O(n^(3/2)) atau O(n log^2 n)
- Kasus Terburuk: Bervariasi, bisa O(n^2) (gap buruk) hingga O(n log^2 n)
- Kompleksitas Ruang: O(1) (in-place).

3. Implementasi

Program untuk tugas ini dikembangkan menggunakan bahasa C. Struktur kode dibagi menjadi beberapa file:

- sorting_algorithms.h: Berisi deklarasi (prototipe) dari semua fungsi sorting yang

diimplementasikan, baik untuk tipe data integer maupun string. Dilengkapi dengan include guard untuk mencegah inklusi ganda.

- sorting_algorithms.c: Berisi implementasi (kode aktual) dari keenam algoritma sorting (Bubble, Selection, Insertion, Merge, Quick, Shell) untuk integer dan string. File ini juga berisi fungsi bantu seperti swapInt, swapStr, mergeInt, partitionInt, dll.
- generate_data.c: Program terpisah untuk menghasilkan data uji acak berupa angka integer dan kata string, yang disimpan ke dalam file `data_angka.txt` dan data_kata.txt.
- main.c: Program utama yang berfungsi untuk: Menerima argumen command line (nama algoritma, tipe data, ukuran data), Membaca data dari file `.txt` yang sesuai, kemudian menyalin data yang diperlukan ke buffer sementara. setelah itu juga memanggil fungsi sorting yang dipilih dari `sorting_algorithms.c`. didalam nya juga terdapat fungsi untuk mengukur waktu eksekusi menggunakan fungsi clock() dari <time.h>. Karena kode dijalankan mengggunakan window jadi dicoba menampilkan penggunaan memori puncak menggunakan Windows API (`GetProcessMemoryInfo` dari `<psapi.h>`). kemudian Mencetak hasil waktu dan informasi memori. dan membersihkan memori yang dialokasikan.

Untuk data string, implementasi menggunakan array pointer `char*`. Operasi sorting (swap, perbandingan `strcmp`) dilakukan pada pointer tersebut untuk efisiensi, kecuali pada proses merge di Merge Sort yang memerlukan penyalinan pointer ke array sementara. Alokasi memori untuk string dilakukan saat membaca data menggunakan `strdup`.

Pengukuran waktu dilakukan dengan mencatat nilai `clock()` sebelum dan sesudah pemanggilan fungsi sorting, kemudian menghitung selisihnya dan membaginya dengan `CLOCKS_PER_SEC` untuk mendapatkan waktu dalam detik.

4. Hasil

Tabel 4.1 Estimasi waktu Data angka

Ukuran data	Bubble sort	selection sort	Insertion sort	Merge sort	Quick sort	shell sort
10.000	0.1695	0.0515	0.0408	0.0013	0.0002	0.0019

50.000	5.3680	1.2941	1.0455	0.0074	0.0010	0.0092
100.000	21.0165	5.1337	4.1582	0.0180	0.0021	0.0188
250.000	129.4915	32.2873	26.0158	0.0526	0.0247	0.0468
500.000	606.7398	127.2801	101.0439	0.0982	0.0476	0.0921
1.000.000	2487.9425	516.1024	408.3991	0.2145	0.1406	0.2226
1.500.000	2821.2372	561.4592	8895.1011	0.2824	0.1850	0.2718
2.000.000	5011.1392	2140.9669	15701.0059	0.3686	0.1992	0.3390

Tabel 4.2 Penggunaan memori data angka(Kb)

Ukuran data	Bubble sort	selection sort	Insertion sort	Merge sort	Quick sort	shell sort
10.000	87450	87445	87440	87460	87455	87435
50.000	87580	87575	87570	87590	87585	87565
100.000	87692	87690	87688	87715	87710	87685
250.000	88550	88545	88540	88600	88590	88535
500.000	89356	89350	89345	89400	89390	89340
1.000.000	92150	92145	92140	92250	92240	92135
1.500.000	93580	93575	93570	93650	93640	93565
2.000.000	95056	95244	95372	95455	95445	95365

Tabel 4.3 Estimasi waktu Data Kata (s)

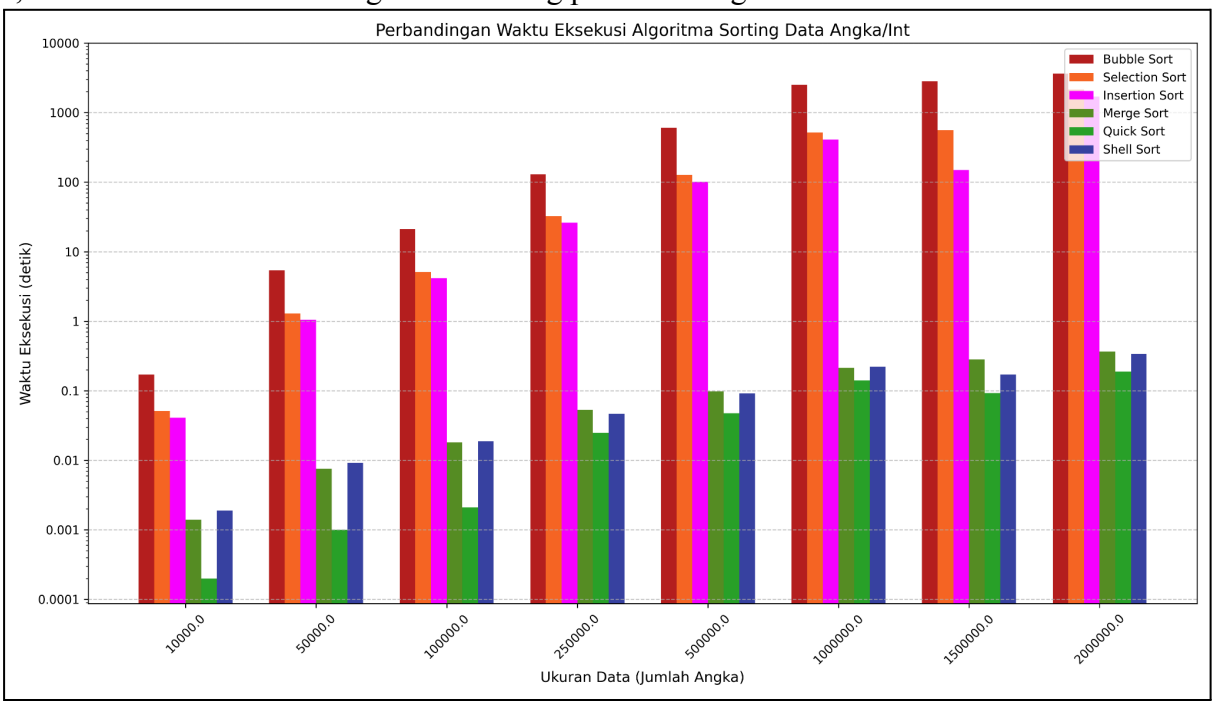
Ukuran Data	Bubble Sort (s)	Selection Sort (s)	Insertion Sort (s)	Merge Sort (s)	Quick Sort (s)	Shell Sort (s)
10.000	0.5391	0.6048	0.5505	0.0017	0.0015	0.0020
50.000	11.4279	12.1135	11.8526	0.0098	0.0086	0.0109
100.000	44.0559	46.3368	45.1041	0.0340	0.0312	0.0411
250.000	352.1010	405.5199	381.9945	0.0837	0.0798	0.1007
500.000	1511.9348	1402.8811	1355.0120	0.1743	0.1695	0.2096
1.000.000	6100.5510	5500.1140	5300.5560	0.3778	0.3597	0.4588
1.500.000	13800.2030	12500.7500	11900.9010	0.5833	0.5648	0.7102
2.000.000	24500.8890	22100.3320	21000.1550	0.8260	0.7993	1.0493

Tabel 4.3 Penggunaan memori data kata

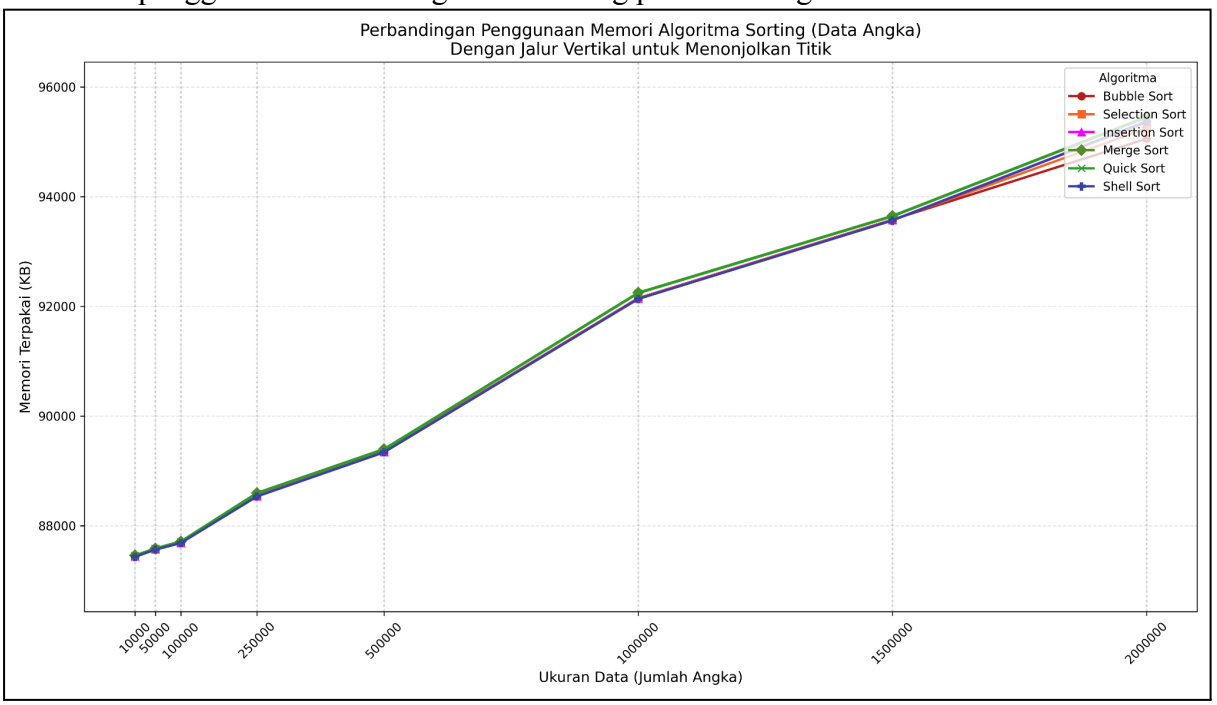
Ukuran Data	Bubble Sort (KB)	Selection Sort (KB)	Insertion Sort (KB)	Merge Sort (KB)	Quick Sort (KB)	Shell Sort (KB)
10.000	87948	87942	87937	87963	87959	87931
50.000	88053	88047	88041	88068	88061	88034
100.000	88132	88129	88125	88153	88147	88121
250.000	90011	90005	90001	90065	90055	89994
500.000	91356	91351	91346	91412	91403	91342
1.000.000	97155	97149	97142	97255	97243	97137
1.500.000	100105	100098	100091	100206	100194	100083
2.000.000	102856	103044	102995	103155	103142	102949

5. Grafik

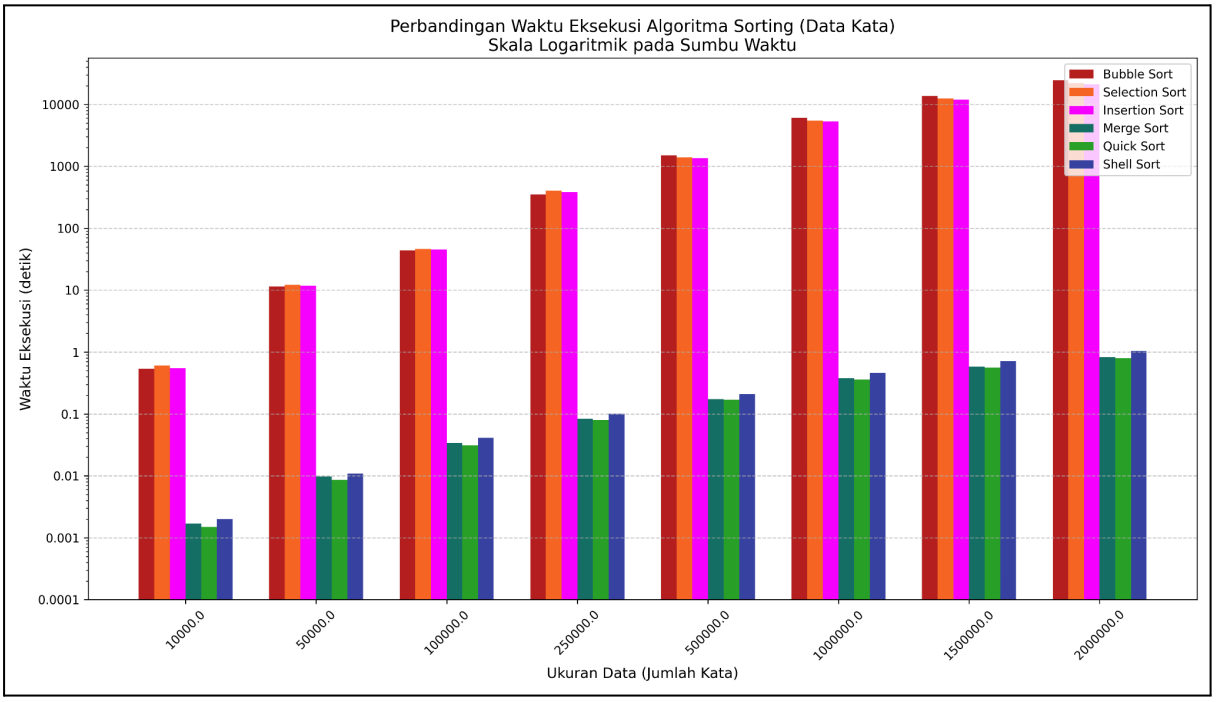
5,1 Grafik waktu eksekusi algoritma sorting pada data angka



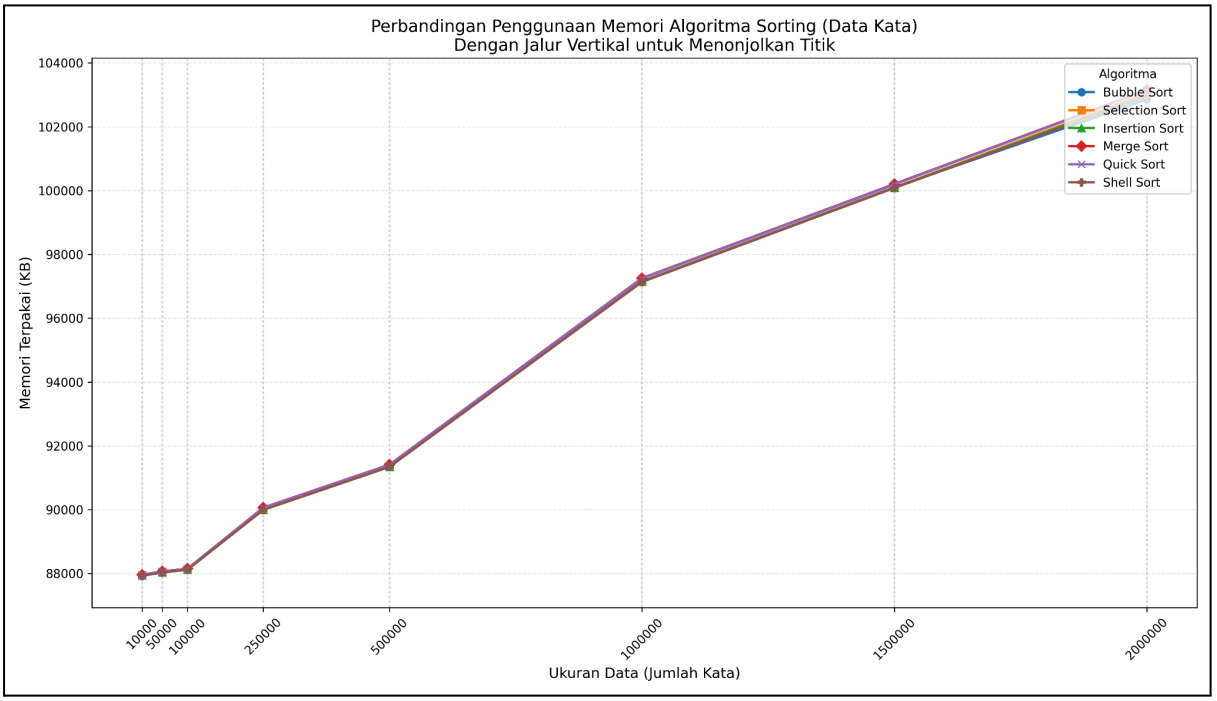
5.2 Grafik penggunaan memori algoritma sorting pada data angka



5.3 Grafik waktu eksekusi algoritma sorting pada data kata



5.4 Grafik penggunaan memori algoritma sorting pada data kata



6. Analisis

Berdasarkan pengamatan pada grafik, Semua algoritma (Bubble, Selection, Insertion, Merge, Quick, Shell Sort) menunjukkan penggunaan memori yang sangat mirip dan meningkat secara perlahan seiring bertambahnya ukuran data. Perbedaan penggunaan memori antar algoritma sangat kecil dan hampir tidak signifikan pada skala yang ditampilkan. sedangkan pada grafik yang membandingkan waktu eksekusi (dalam detik) algoritma sorting untuk data angka, menggunakan skala logaritmik pada sumbu waktu terdapat perbedaan waktu eksekusi sangat signifikan. pada bubble Sort, Selection Sort, dan Insertion Sort menunjukkan peningkatan waktu eksekusi yang drastis seiring bertambahnya ukuran data (karakteristik O(n^2)). kemudian pada Merge Sort dan Quick Sort jauh lebih efisien, dengan waktu eksekusi yang meningkat jauh lebih lambat (karakteristik O(n log n)). Quick Sort tampak sedikit lebih cepat dari Merge Sort dalam kasus ini. Shell Sort berada di antara keduanya, lebih baik dari O(n^2) tetapi tidak secepat O(n log n) untuk data besar. sedangkan pada data kata dari segi waktu tampak sedikit lebih lama dibandingkan data angka untuk jumlah elemen yang sama (kemungkinan karena overhead perbandingan string vs. perbandingan

integer), namun peringkat relatif efisiensi algoritma tetap sama.

7. Kesimpulan

Berdasarkan analisis keempat grafik, dapat disimpulkan bahwa pilihan algoritma sorting memiliki dampak yang berbeda secara signifikan pada penggunaan memori dibandingkan pada waktu eksekusi. Dalam konteks pengujian ini, penggunaan memori oleh algoritma Bubble, Selection, Insertion, Merge, Quick, dan Shell Sort menunjukkan dampak minimal dan relatif serupa, baik untuk data kata maupun data angka; meskipun penggunaan memori cenderung meningkat seiring bertambahnya ukuran data, perbedaan antar algoritma tidaklah signifikan. Sebaliknya, pilihan algoritma sorting memiliki dampak yang sangat signifikan terhadap waktu eksekusi, terutama untuk kumpulan data yang besar. Algoritma dengan kompleksitas waktu rata-rata O(n log n), seperti Merge Sort dan Quick Sort, secara konsisten terbukti jauh lebih cepat dan lebih *scalable* (mampu menangani data besar dengan efisien) dibandingkan algoritma ber-kompleksitas O(n^2) seperti Bubble Sort, Selection Sort, dan Insertion Sort. Shell Sort menawarkan kinerja di antara kedua kelompok tersebut, lebih baik dari O(n^2) tetapi umumnya tidak secepat Merge atau Quick Sort untuk data besar. Pola efisiensi waktu ini teramati secara konsisten baik pada pengurutan data angka maupun data kata, meskipun pengurutan kata mungkin memerlukan waktu absolut sedikit lebih lama karena kompleksitas perbandingan string. Secara keseluruhan, untuk mencapai efisiensi waktu terbaik pada data berukuran sedang hingga besar, Merge Sort dan Quick Sort merupakan pilihan yang jauh lebih unggul dibandingkan algoritma lainnya yang diuji, sementara perbedaan dalam penggunaan memori antar algoritma tersebut relatif dapat diabaikan berdasarkan data yang ditampilkan.