ANALISIS EFISIENSI MEMORI DAN WAKTU EKSEKUSI ALGORITMA *QR SORT* DAN *BUBBLE SORT* DALAM C++

Adi Handika[1)], Hafidzah Azzahrani[2)], Rizqi Karima[3)], Mualim[4)], Imam Prayogo Pujiono[5)]
[1,2,3,4]Prodi Bisnis Digital, Fakultas Ekonomi dan Bisnis Islam, UIN K.H. Abdurrahman Wahid
[5]Prodi Informatika, Fakultas Ekonomi dan Bisnis Islam, UIN K.H. Abdurrahman Wahid

Correspondence author: A.Handika, adi.handika25004@mhs.uingusdur.ac.id, Pekalongan, Indonesia

Abstract

Sorting is a fundamental process in computer science that significantly impacts data management and computational efficiency. This study aims to compare the performance of two sorting algorithms, QR Sort and Bubble sort, implemented in C++. The comparison analyzes and evaluates the two algorithms in terms of execution time and memory usage efficiency across datasets of varying sizes. This study uses a quantitative experimental method, with three dataset sizes: 100, 1,000, and 10,000 integer elements, which are run repeatedly under identical conditions to ensure consistent results. The experiments measure the average execution time (in microseconds) and estimated memory usage (in kilobytes) for each algorithm. The results show that QR Sort, which applies a non-comparative quotient–remainder approach, performs significantly faster than Bubble sort, which is comparative, especially as the data size increases. On large datasets, QR Sort outperforms Bubble sort by more than 100 times in execution time. This speed increase is accompanied by higher memory consumption, as QR Sort requires additional structures such as a bucket and counting arrays, whereas Bubble sort uses only minimal memory. Overall, these findings confirm that QR Sort is better suited for large-scale data processing where speed is a priority. At the same time, Bubble sort remains more efficient for small datasets or educational purposes due to its simplicity and low memory requirements. This study emphasizes the need to choose an appropriate sorting algorithm based on dataset characteristics and available system resources.

Keywords: sorting, execution time, memory usage efficiency, C++

Abstrak

Pengurutan (*Sorting*) adalah proses dasar dalam ilmu komputer yang memiliki pengaruh besar terhadap pengelolaan data dan efisiensi komputasi. Penelitian ini bertujuan untuk membandingkan kinerja dua algoritma pengurutan, yaitu *QR Sort* dan *Bubble sort*, yang diimplementasikan menggunakan bahasa pemrograman c++. Perbandingan dilakukan untuk menganalisis dan mengevaluasi kedua algoritma tersebut dari segi efisiensi waktu eksekusi dan penggunaan memori ketika diterapkan pada kumpulan data dengan ukuran yang berbeda-beda. Penelitian ini menggunakan metode eksperimen kuantitatif, dengan tiga skala dataset: 100, 1.000, dan 10.000 elemen bilangan bulat, yang dijalankan berulang kali dalam kondisi identik untuk memastikan konsistensi hasil. Percobaan mengukur rata-rata waktu eksekusi (dalam mikrodetik) dan perkiraan penggunaan memori (dalam kilobyte) untuk masing-masing algoritma. Hasil penelitian menunjukkan bahwa *QR Sort*,

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

yang menerapkan pendekatan *non-comparative quotient–remainder*, bekerja jauh lebih cepat dibandingkan *Bubble sort* yang bersifat comparative, terutama saat ukuran data meningkat. pada dataset besar, *QR Sort* mengungguli *Bubble sort* dengan kecepatan lebih dari 100 kali lipat dalam waktu eksekusi. Peningkatan kecepatan ini disertai dengan konsumsi memori yang lebih tinggi karena *QR Sort* memerlukan struktur tambahan seperti *bucket* array dan counting array, sedangkan *Bubble sort* hanya menggunakan memori dalam jumlah minimal. Secara keseluruhan, temuan ini menegaskan bahwa *QR Sort* lebih cocok untuk pemrosesan data berskala besar di mana kecepatan menjadi prioritas, sedangkan *Bubble sort* tetap lebih efisien untuk dataset kecil atau keperluan edukatif karena kesederhanaannya dan kebutuhan memori yang rendah. Penelitian ini menekankan perlunya memilih algoritma pengurutan yang sesuai berdasarkan karakteristik dataset dan sumber daya sistem yang tersedia.

Kata Kunci: algotirma pengurutan, efisiensi memori, waktu eksekusi, C++

A. PENDAHULUAN

Sorting merupakan salah satu proses mendasar dalam ilmu komputer yang berperan penting dalam pengelolaan, pencarian, dan analisis data. Algoritma pengurutan yang efisien dapat meningkatkan kinerja berbagai aplikasi, mulai dari sistem basis data, *search engine*, hingga sistem rekomendasi dan analitik data berskala besar (Astuti, 2023). Dalam konteks pemrograman modern, khususnya di C++, pemilihan algoritma pengurutan yang tepat menjadi krusial karena berdampak langsung terhadap waktu eksekusi dan penggunaan sumber daya sistem (Pujiono et al., 2025).

Kompleksitas waktu dan ruang (memori) merupakan dua indikator utama yang digunakan untuk mengevaluasi kinerja suatu algoritma pengurutan (Ariza et al., 2025). Pemilihan algoritma yang tepat umumnya mempertimbangkan ukuran dataset, distribusi data (acak, hampir terurut, terurut terbalik, dan sebagainya), serta keterbatasan sumber daya sistem (Bushman et al., 2024; Musyaffa et al., 2025).

Berbagai algoritma *sorting* telah dikembangkan, mulai dari algoritma komparatif sederhana seperti *Bubble sort*, *Insertion Sort*, dan *Selection Sort*, hingga algoritma yang lebih efisien secara asimtotik seperti *Quick Sort*, *Merge Sort*, dan *Heap Sort*. Di sisi lain, terdapat pula algoritma non komparatif seperti *Counting Sort*, *Radix Sort*, dan *Bucket sort* yang memanfaatkan karakteristik khusus data untuk mencapai kinerja mendekati linear (Ali et al., 2025; Marcellino et al., 2021). *Bubble sort*, meskipun memiliki kompleksitas waktu $O(n^2)$ dan dikenal tidak efisien untuk dataset besar, masih banyak digunakan dalam konteks edukatif karena implementasinya yang sederhana dan mudah dipahami (Barokah et al., 2025).

Bubble sort termasuk keluarga algoritma pengurutan komparatif sederhana dengan kompleksitas waktu $O(n^2)$ (Gustina et al., 2025; Musyaffa et al., 2025). Algoritma ini bekerja dengan cara membandingkan pasangan elemen yang berdekatan dan menukarnya apabila berada pada urutan yang salah. Meskipun mudah diimplementasikan dan sering digunakan sebagai contoh pengajaran dasar struktur data dan algoritma, kinerja *Bubble sort* menurun drastis ketika diaplikasikan pada dataset berukuran besar (Barokah et al., 2025; Mahrozi & Faisal, 2023). Sejumlah penelitian menunjukkan bahwa *Bubble sort* cenderung mempunyai waktu eksekusi paling lambat di antara algoritma *sorting* lain, meski tetap unggul dari sisi kesederhanaan dan konsumsi memori yang rendah (Ali et al., 2025).

Sebaliknya, algoritma non komparatif seperti *Counting Sort*, *Radix Sort*, dan *Bucket sort* memanfaatkan karakteristik khusus data untuk mengurangi jumlah operasi perbandingan dan mencapai kompleksitas waktu mendekati O(n) (Fenyi et al., 2020; Marcellino et al., 2021). *QR Sort* (*Quotient – Remainder* Sort) yang diperkenalkan oleh Bushman (Bushman et al., 2024) termasuk dalam kategori ini. *QR Sort* memetakan setiap elemen ke dalam *bucket* berdasarkan hasil bagi (*quotient*) dan sisa bagi (*remainder*) terhadap suatu nilai pembagi tertentu (*divisor*). Pendekatan ini memungkinkan proses pengurutan dilakukan melalui transformasi aritmetika sederhana tanpa perlu membandingkan elemen satu per satu.

Pendekatan ini memungkinkan pengurangan jumlah operasi perbandingan secara signifikan dan secara teoretis menawarkan kompleksitas waktu yang lebih baik dibandingkan algoritma kuadratik tradisional. Studi awal menunjukkan bahwa *QR Sort* berpotensi memberikan peningkatan kecepatan yang substansial pada data numerik berukuran besar (Bushman et al., 2024).

Sejumlah penelitian sebelumnya telah mengkaji kinerja algoritma *sorting* dari sisi waktu eksekusi dan penggunaan memori. (Ali et al., 2025), misalnya, melakukan analisis komparatif terhadap delapan algoritma *sorting* dengan fokus pada efisiensi komputasi di C++. Penelitian lain membahas perbandingan algoritma *sorting* lanjutan, termasuk *Quick Sort*, *Heap Sort*, *Merge Sort*, dan *Radix Sort*, terutama dari perspektif waktu eksekusi (Marcellino et al., 2021; Musyaffa et al., 2025). Berbagai studi juga telah membahas implementasi berbasis C++ dan pemanfaatan *Standard Template Library* (STL) (Musyaffa et al., 2025; Zahwa et al., 2025). Namun, kajian yang secara eksplisit memfokuskan pada perbandingan *QR Sort* dan *Bubble sort* dari sisi *time space trade off* dalam implementasi C++ masih terbatas. Hal ini membuka ruang penelitian untuk mengevaluasi seberapa jauh *QR Sort* mampu memberikan peningkatan kinerja waktu eksekusi serta implikasinya terhadap konsumsi memori ketika dibandingkan dengan algoritma kuadratik sederhana seperti *Bubble sort*

Secara spesifik, kontribusi utama penelitian ini adalah sebagai berikut:

1. menyajikan evaluasi empiris algoritma *QR Sort* berbasis C++ dan membandingkannya secara langsung dengan *Bubble sort* pada tiga skala data (100, 1.000, dan 10.000 elemen);
2. mengukur dan menganalisis *trade off* antara efisiensi waktu eksekusi dan penggunaan memori pada kedua algoritma menggunakan pendekatan eksperimen terkontrol;
3. mengintegrasikan hasil empiris dengan analisis kompleksitas teoretis untuk memberikan rekomendasi praktis pemilihan algoritma pengurutan pada aplikasi C++ modern.

Berdasarkan kondisi tersebut, penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja algoritma *QR Sort* dan *Bubble sort* dalam bahasa pemrograman C++ dengan fokus pada efisiensi waktu eksekusi dan penggunaan memori pada berbagai ukuran dataset integer acak. Dengan demikian, penelitian ini diharapkan dapat memberikan gambaran yang lebih komprehensif mengenai *trade off* (waktu dan memori) antara algoritma non komparatif dan komparatif pada konteks implementasi nyata.

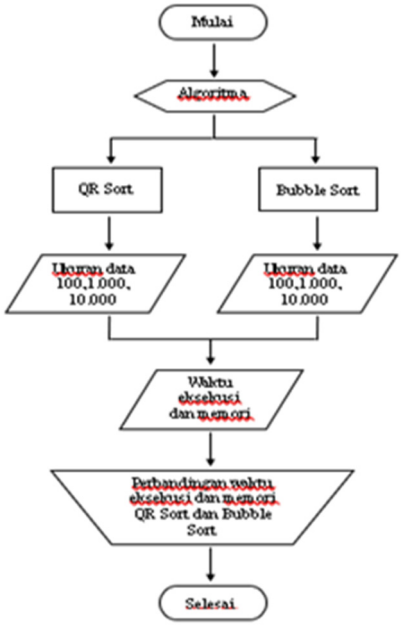
B. METODE PENELITIAN

Penelitian ini dilakukan dengan menggunakan Bahasa pemrograman C++ untuk menjalankan sebuah kode program. Pengujian dijalankan pada laptop dengan spesifikasi : Processor AMD A4-9115, SSD 500 GB HDD, Memori RAM 4 GB, Sistem operasi Windows 10, Merk Laptop ASUS E401B

Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen komparatif untuk mengetahui efisiensi memori dan waktu komputasi antara algoritma pengurutan data (*QR Sort* dan *Bubble sort*). Metode Eksperimen digunakan karena adanya

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

pengujian langsung terhadap kinerja algoritma sehingga menghasilkan data yang terukur. Penelitian dilakukan guna mengevaluasi kinerja algoritma secara sistematis, dengan focus pada dua kriteria: penggunaan memori dan waktu komputasi. Untuk memperjelas proses, tahap dalam penelitian meliputi tahap persiapan dataset, implementasi algoritma, pengujian kinerja, pengumpulan data, dan analisis komparatif. Tahap eksperimen dimulai dengan membuat dataset yang dibagi menjadi tiga ukuran yaitu : kecil (100 elemen), sedang (1000 elemen), dan besar (10.000 elemen) untuk mengukur skalabilitas algoritma. *Flowchart* alur logika yang digunakan dapat dilihat dibawah ini:



Gambar 1. *Flowchart* Alur Logika

Selanjutnya, kedua algoritma diimplementasikan dalam Bahasa pemrograman C++ dengan kode program ditulis dan dijalankan melalui *Codebloks*. Pengujian kinerja dilakukan dengan menjalankan setiap algoritma pada ketiga ukuran dataset, mengukur waktu komputasi dan penggunaa memori yang digunakan selama proses pengurutan. Pengujian dilakukan melalui eksperimen berulang setiap data set untuk memastikan konsistensi hasil suatu program. Data yang dikumpulkan berupa waktu komputasi (satuan mikrodetik) dan penggunaan memori selama proses berlangsung. Data hasil pengujian dikumpulkan dan dibandingkan untuk menentukan algoritma dengan efisiensi terbaik berdasarkan kedua kriteria tersebut. Hasil pegujian ini diharapkan memberikan gambaran kinerja nyata dari masing masing kedua algoritma pengurutan dalam skenario dataset berskala kecil,sedang hingga besar.

C. HASIL DAN PEMBAHASAN

Penelitian ini dilakukan dengan melakukan simulasi pengujian terhadap dua algoritma pengurutan data, yaitu *QR Sort* dan *Bubble sort*, menggunakan bahasa pemrograman C++. Pengujian dilakukan sebanyak tiga kali percobaan dengan variasi ukuran data sebanyak 100, 1.000, dan 10.000 elemen, menggunakan pendekatan pengukuran waktu eksekusi (dalam mikrodetik) dan estimasi penggunaan memori (dalam *kilobyte*). Hasil pengujian ditunjukkan pada Tabel 1 dan Tabel 2, sebagaimana diperoleh dari program yang dijalankan menggunakan CODE::BLOCKS pada lingkungan eksperimen yang sama tanpa optimasi tambahan.

Tabel 1. Hasil simulasi waktu eksekusi dari kedua algoritma *sorting* dengan 3 kali percobaan *compile*

Ukuran Data	Percobaan Ke	*QR Sort*(µs)	*Bubble sort* (µs)
100	1	314	93
	2	406	100
	3	199	98
Rata-rata		339,6	97
1.000	1	1.684	10.378
	2	1.313	13.071
	3	1.389	13.141
Rata-rata		1.465	11.197
10.000	1	6.414	1.135.375
	2	4.481	1.148.347
	3	3.801	1.093.749
Rata-rata		11.376	1.135.711,67

Hasil dari Tabel 1. menunjukkan bahwa waktu eksekusi *QR Sort* meningkat secara linear terhadap pertambahan ukuran data, sedangkan *Bubble sort* meningkat secara kuadratik. Pada ukuran data besar (1000 elemen), *QR Sort* hanya membutuhkan rata-rata 1.465 mikrodetik, sedangkan *Bubble sort* memerlukan waktu 11.197 mikrodetik yang artinya *QR Sort* dapat bekerja 8 kali lebih cepat dari *Bubble sort* dan pada ukuran data besar (10000 elemen), *QR Sort* hanya membutuhkan rata-rata 11.376 mikrodetik, sedangkan *Bubble sort* memerlukan waktu lebih dari satu juta mikrodetik yang artinya *QR Sort* dapat bekerja ±100 kali lebih cepat dari *Bubble sort*. Hal ini mengindikasikan efisiensi signifikan *QR Sort* dalam menangani dataset berukuran besar.

Tabel 2. Hasil pengujian perkiraan pemakaian memori untuk kedua metode *sorting* dengan 3 kali percobaan

Ukuran Data	Percobaan Ke	*QR Sort* (KB)	*Bubble sort* (KB)
100	1	64,9	0,4
	2	68,7	0,4
	3	64,9	0,4
Rata-rata		66,16	0,4
1.000	1	304,6	3,93
	2	187,4	3,93
	3	191,8	3,93
Rata-rata		194,9	3,93
10.000	1	489,5	39,09
	2	486,7	39,09
	3	488,3	39,09
Rata-rata		488,16	39,09

Berdasarkan Tabel 2 yang menampilkan hasil simulasi pemakaian memori dari algoritma *QR Sort* dan *Bubble sort*, dapat dijelaskan bahwa terjadi perbedaan yang cukup signifikan dalam kebutuhan ruang penyimpanan di antara keduanya. Pada ukuran data 100 elemen, *QR Sort* menggunakan rata-rata memori sebesar 66,16 KB, sedangkan *Bubble sort* hanya membutuhkan sekitar 0,4 KB. Hal ini menunjukkan bahwa sejak ukuran data terkecil, *QR Sort* sudah memerlukan alokasi memori yang jauh lebih besar karena melibatkan pembentukan struktur data tambahan seperti *bucket* array dan counting array di setiap tahap proses pengurutan. Ketika ukuran data meningkat menjadi 1.000 elemen, kebutuhan memori *QR Sort* naik menjadi rata-rata 194,9 KB, sementara *Bubble sort* hanya meningkat sedikit menjadi 3,93 KB. Peningkatan ini mengindikasikan bahwa *QR Sort* memiliki pola pertumbuhan penggunaan memori yang bersifat hampir linear terhadap jumlah data, sedangkan *Bubble sort* cenderung lambat meningkat karena hanya bekerja dengan satu struktur vektor utama tanpa tambahan alokasi dinamis. Pada ukuran data terbesar, yaitu 10.000 elemen, *QR Sort* memerlukan rata-rata memori sebesar 488,16 KB, sedangkan *Bubble sort* sebesar 39,09 KB. Walaupun *QR Sort* tampak jauh lebih boros dalam penggunaan memori, konsumsi ruang sebesar 488 KB masih tergolong kecil untuk sistem komputer modern dan sebanding dengan keuntungan besar yang diberikan dari segi kecepatan eksekusi. Dengan kata lain, *QR Sort* melakukan kompromi antara waktu dan ruang (time space, *trade off*), di mana peningkatan konsumsi memori dibayar dengan performa waktu proses yang jauh lebih efisien.

Dari sisi kompleksitas teoritis, *QR Sort* dan *Bubble sort* memiliki karakteristik yang sangat berbeda karena keduanya mewakili dua paradigma pengurutan yang kontras: *non-comparative* dan comparative. *Bubble sort* bekerja dengan pendekatan pairwise comparison, yaitu membandingkan setiap elemen dengan elemen berikutnya dan menukarnya bila urutannya salah (Mahrozi & Faisal, 2023). Hal ini menyebabkan kompleksitas waktu *Bubble sort* bersifat kuadratik, yaitu O(n2). Setiap iterasi luar membutuhkan hingga n-1 perbandingan, dan iterasi dalam juga mendekati n-1 kali, sehingga jumlah total operasi mendekati n(n-1)/2. Akibatnya, peningkatan ukuran data

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

akan menyebabkan waktu eksekusi meningkat secara eksponensial, seperti yang tampak pada hasil eksperimen ketika ukuran data naik dari 1.000 ke 10.000 elemen. Sebaliknya, *QR Sort* (*Quotient–Remainder* Sort) dirancang dengan konsep *non-comparative sorting*, yang berarti proses pengurutan tidak dilakukan dengan membandingkan nilai satu per satu (Bushman et al., 2024). *QR Sort* memanfaatkan transformasi aritmetika sederhana berupa pembagian dan sisa bagi untuk mengelompokkan elemen ke dalam *bucket* berdasarkan rentang nilai tertentu. Karena tidak bergantung pada operasi perbandingan, kompleksitas waktu teoritisnya dapat didekati dengan O(n+k), di mana k adalah jumlah *bucket* yang terbentuk. Dalam praktiknya, karena kjauh lebih kecil dari ndan proporsional terhadap akar dari rentang nilai, kompleksitas totalnya mendekati O(n), atau bersifat linear terhadap ukuran data..

Kinerja empiris hasil eksperimen mendukung analisis teoritis tersebut. Ketika jumlah elemen meningkat sepuluh kali lipat (dari 1.000 ke 10.000), waktu eksekusi *QR Sort* hanya bertambah sekitar tiga kali lipat, sedangkan *Bubble sort* meningkat hampir seratus kali lipat. Pola ini menunjukkan bahwa *QR Sort* memang tumbuh secara linear (proporsional terhadap jumlah data), sementara *Bubble sort* meningkat secara kuadratik. Hal ini juga sesuai dengan prediksi analisis asimtotik bahwa metode berbasis perbandingan memiliki batas bawah O(nlog n), sedangkan metode non-komparatif seperti *QR Sort* bisa mencapai kinerja mendekati O(n). Dari sisi kompleksitas ruang (space complexity), hasilnya juga memperkuat teori yang ada. *Bubble sort* memiliki kebutuhan ruang tambahan yang minimal, karena seluruh operasi dilakukan langsung pada array data tanpa alokasi struktur tambahan besar. Secara teoritis, kompleksitas ruangnya adalah O(1) atau konstan, karena hanya menggunakan beberapa variabel tambahan untuk menyimpan nilai sementara dalam proses pertukaran data.

Sebaliknya, *QR Sort* membutuhkan alokasi memori tambahan untuk struktur *bucket* dan array penghitung di setiap *bucket*. Jika jumlah *bucket* yang terbentuk adalah k, maka total ruang tambahan yang dibutuhkan dapat dinyatakan dengan O(n+k). Dalam implementasi C++ yang digunakan, setiap *bucket* direpresentasikan sebagai vector<int>, dan masing-masing *bucket* menyimpan sejumlah *remainder* yang bervariasi sesuai distribusi data acak. Akibatnya, konsumsi memori *QR Sort* meningkat hampir secara linear terhadap pertambahan jumlah data. Hasil empiris juga menunjukkan tren yang sejalan dengan analisis tersebut, di mana penggunaan memori *QR Sort* meningkat dari sekitar ±66 KB (100 elemen) menjadi hampir ±500 KB (10.000 elemen).

Perbandingan ini menegaskan adanya hubungan *trade off* antara waktu dan ruang. *Bubble sort* unggul secara signifikan dalam efisiensi ruang, tetapi membayar harga mahal dalam waktu eksekusi. Sebaliknya, *QR Sort* mengorbankan sebagian memori tambahan untuk memperoleh efisiensi waktu yang jauh lebih tinggi. Dalam konteks komputasi modern yang memiliki kapasitas memori besar, pertukaran seperti ini dianggap menguntungkan karena kecepatan pemrosesan sering kali menjadi faktor yang lebih penting daripada hemat ruang.

Hasil pengujian ini memiliki sejumlah implikasi penting, baik secara akademik maupun praktis. Dari sisi akademik, penelitian ini memperkuat pemahaman tentang hubungan antara paradigma algoritmik dan performa nyata dalam konteks pengolahan data. *QR Sort*, sebagai algoritma *non-comparative*, membuktikan bahwa pendekatan alternatif di luar metode perbandingan klasik seperti *Bubble sort*, *Insertion Sort*, atau *Selection Sort* dapat menghasilkan peningkatan performa signifikan. Ini membuka ruang bagi penelitian lanjutan dalam bidang integer *sorting* dan *non-comparative algorithm design*, yang relevan untuk aplikasi modern seperti pengolahan data numerik dalam *machine learning*, *data mining*, dan *big data*

processing (Bushman et al., 2024; Dong et al., 2024; Fenyi et al., 2020; Werdiningsih et al., 2022). Selain itu, hasil ini juga menunjukkan pentingnya pemilihan algoritma berdasarkan konteks kebutuhan sistem. Pada lingkungan dengan keterbatasan sumber daya memori, *Bubble sort* atau algoritma sederhana lainnya masih dapat dipertimbangkan karena mudah diimplementasikan dan hemat ruang. Namun, pada sistem dengan sumber daya yang memadai dan tuntutan waktu eksekusi tinggi seperti sistem real-time, basis data besar, atau perangkat analitik *QR Sort* jauh lebih layak digunakan karena skalabilitas dan efisiensi temporalnya yang unggul.

Secara praktis, penelitian ini juga menunjukkan bahwa tidak semua algoritma yang sederhana secara logika cocok untuk aplikasi dunia nyata. *Bubble sort* sering digunakan sebagai contoh pembelajaran dasar algoritma pengurutan karena mudah dipahami dan diimplementasikan, tetapi performanya terbatas untuk dataset berukuran kecil. Ketika diaplikasikan pada skala besar, biaya komputasinya menjadi tidak efisien dan dapat menyebabkan bottleneck pada sistem. Di sisi lain, *QR Sort* menawarkan alternatif yang efisien dengan konsep matematis sederhana, yakni pemanfaatan pembagian dan sisa bagi untuk mengklasifikasikan data. Pendekatan ini relevan untuk sistem yang mengutamakan waktu respons cepat dengan ukuran data besar, seperti proses batch *sorting* di server atau algoritma penyortiran tahap awal dalam sistem basis data (Utami & Astuti, 2024).

Implikasi lainnya adalah bahwa desain algoritma modern perlu memperhitungkan keseimbangan antara efisiensi waktu dan efisiensi ruang, bukan hanya salah satunya. Dalam banyak kasus nyata, algoritma yang lebih cepat meskipun sedikit lebih boros memori justru memberikan keuntungan yang lebih besar karena mempercepat keseluruhan pipeline pemrosesan data. Prinsip inilah yang ditunjukkan oleh *QR Sort* dalam penelitian ini: peningkatan konsumsi memori relatif kecil dibandingkan dengan peningkatan performa waktu yang diperoleh. Lebih lanjut, hasil penelitian ini juga dapat dijadikan dasar untuk pengembangan algoritma *QR Sort* yang lebih optimal. Misalnya, dengan menyesuaikan parameter pembagi dsecara adaptif berdasarkan distribusi data yang masuk, sehingga *bucket* yang terbentuk lebih seimbang dan meminimalkan jumlah *remainder* yang tidak merata. Pendekatan adaptif semacam ini berpotensi mengurangi konsumsi memori sekaligus mempertahankan efisiensi waktu eksekusi (Bushman et al., 2024). Selain itu, *QR Sort* dapat dikembangkan lebih lanjut untuk mendukung pengurutan tipe data lain seperti *floating point numbers* atau *custom data structures* dengan menambahkan mekanisme normalisasi nilai sebelum proses pembagian.

Dari sisi metodologi, hasil eksperimen juga menunjukkan pentingnya pengujian empiris dengan variasi ukuran data yang luas. Analisis teoritis sering kali hanya memberikan gambaran ideal, tetapi implementasi nyata dapat dipengaruhi oleh faktor-faktor lain seperti efisiensi manajemen memori, arsitektur prosesor, dan optimasi kompilator (Harsono et al., 2022; Rizka, 2022). Karena itu, pendekatan seperti yang dilakukan dalam penelitian ini menggabungkan eksperimen terukur dengan analisis matematis sehingga memberikan hasil yang lebih komprehensif dan dapat dijadikan acuan dalam evaluasi performa algoritma lain di masa mendatang. Terakhir, penelitian ini menegaskan bahwa efisiensi algoritma bukan hanya bergantung pada kecepatan proses *sorting* itu sendiri, tetapi juga pada bagaimana algoritma tersebut memanfaatkan sumber daya komputer secara keseluruhan. *QR Sort* yang dirancang dengan prinsip arithmetic mapping membuktikan bahwa operasi sederhana dapat menghasilkan penghematan waktu yang besar tanpa memerlukan logika perbandingan yang kompleks (Bushman et al., 2024). Hal ini menunjukkan arah baru dalam riset optimasi algoritma yang berorientasi pada efisiensi struktur dan bukan semata pada jumlah operasi logis.

D. PENUTUP

Berdasarkan hasil analisis komputasi yang telah dilakukan, dapat disimpulkan bahwa algoritma *QR Sort* memiliki keunggulan utama pada efisiensi waktu eksekusi, terutama ketika menangani data berukuran besar. Kompleksitas algoritmiknya yang mendekati linear menjadikan *QR Sort* lebih stabil dan skalabel dibandingkan *Bubble sort* yang bersifat kuadratik. Walaupun *QR Sort* membutuhkan tambahan memori untuk penyimpanan struktur bantu, peningkatan tersebut sebanding dengan keuntungan waktu proses yang signifikan. Dengan demikian, *QR Sort* lebih direkomendasikan untuk aplikasi yang membutuhkan kecepatan tinggi dalam pemrosesan data berukuran besar, sementara *Bubble sort* tetap relevan digunakan untuk pembelajaran dasar atau sistem dengan keterbatasan memori.

Adapun saran untuk penelitian selanjutnya yaitu agar dilakukan eksplorasi terhadap pengembangan *QR Sort* dengan pendekatan adaptif, misalnya dengan menyesuaikan parameter pembagi dan jumlah *bucket* secara dinamis berdasarkan karakteristik distribusi data. Selain itu, penelitian komparatif dengan algoritma non comparative lainnya seperti Radix Sort, Counting Sort, dan *Bucket sort* juga disarankan untuk memperluas perspektif performa *QR Sort* di antara algoritma sejenis. Uji empiris terhadap pola data yang lebih bervariasi juga perlu dilakukan untuk mengukur kestabilan algoritma dalam berbagai skenario praktis. Melalui pengembangan tersebut, diharapkan *QR Sort* dapat dioptimalkan menjadi algoritma yang tidak hanya cepat, tetapi juga lebih efisien dalam penggunaan sumber daya komputasi.

E. DAFTAR PUSTAKA

Ali, M. I., Fardiarsyah, R. D., Shodik, L., Kinanti, F. Z. D., & Pujiono, I. P. (2025). Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++. *LogicLink*, *2*(1), 1–17. https://doi.org/10.28918/logiclink.v2i1.10868

Ariza, S. F., Majid, A., Himawan, I., Ardhiartha P.U., S., & Pujiono, I. P. (2025). Studi Perbandingan Algoritma Pencarian Binary, Jump, Interpolation, dan Fibonacci: Efisiensi Memori dan Waktu Eksekusi. *JATI : Jurnal Mahasiswa Teknik Informatika*, *9*(4), 7227–7234. https://doi.org/10.36040/jati.v9i4.14360

Astuti, Y. A. (2023). Analisis Pengujian Data Algoritma Bubble Sort. *REMIK: Riset Dan E-Jurnal Manajemen Informatika Komputer*, *7*(3), 1413–1420. https://doi.org/10.33395/remik.v7i3.12594

Barokah, M. R. S., Saputra, T., & Sutabri, T. (2025). Perbandingan Kinerja Algoritma Bubble Sort dan Insertion Sort dalam Pengurutan Data Penjualan UMKM. *MIFORTEKH : Jurnal Manajemen Informatika & Teknologi*, *5*(1), 184–195. https://doi.org/10.51903/h0k3zg44

Bushman, R. T., Tebcherani, T. M., & Yasin, A. S. (2024). *QR Sort: A Novel Non-Comparative Sorting Algorithm* (2411.07526; Data Structures and Algorithms). https://doi.org/10.48550/arXiv.2411.07526

Dong, X., Dhulipala, L., Gu, Y., & Sun, Y. (2024). *Parallel Integer Sort: Theory and Practice* (Data Structures and Algorithms). https://doi.org/10.48550/arXiv.2401.00710

Fenyi, A., Fosu, M., & Appiah, B. (2020). Comparative Analysis of Comparison and Non Comparison based Sorting Algorithms. *International Journal of Computer Applications*, *175*, 22–25. https://doi.org/10.5120/ijca2020920813

Gustina, Ahadi, A. H., Aminuddin, F. H., & Syawal, M. F. (2025). Performance

Analysis of Bubble Sort, Selection Sort, and Insertion Sort Algorithms Using Python on Student Data. *JIPTI : Jurnal Inovasi Pendidikan Dan Teknologi Informasi*, *6*(2), 529–536. https://doi.org/10.52060/jipti.v6i2.3792

Harsono, C. A., Shalekha, N. F., & Berliana, R. Y. (2022). Strategi Efisien Manajemen Memori Untuk Meningkatkan Kinerja Sistem Operasi. *SINTESIA: Jurnal Sistem Dan Teknologi Informasi Indonesia*, *2*(1), 12–16. https://journal.unj.ac.id/unj/index.php/SINTESIA/article/view/39413

Mahrozi, N., & Faisal, M. (2023). Analisis Perbandingan Kecepatan Algoritma Selection Sort dan Bubble Sort. *Scientica: Jurnal Ilmiah Sains Dan Teknologi*, *1*(2), 89–98. https://doi.org/10.572349/scientica.v1i2.209

Marcellino, Pratama, D. W., Suntiarko, S. S., & Margi, K. (2021). Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage. *1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, 154–160. https://doi.org/10.1109/ICCSAI53272.2021.9609715

Musyaffa, M. Z., Raharjo, K., Faiz, M., Ammarulloh, S., & Pujiono, I. P. (2025). Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algoritma Pengurutan Tradisional Menggunakan Python. *JEIS: Jurnal Elektro Dan Informatika Swadharma*, *5*(2), 72–81. https://doi.org/10.56486/jeis.vol5no2.785

Pujiono, I. P., Rachmawanto, E. H., & Winarsih, N. A. S. (2025). Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis. *JAMIKA : Jurnal Manajemen Informatika*, *15*(1), 47–59. https://doi.org/10.34010/jamika.v15i1.13230

Rizka, A. (2022). *Organisasi dan Arsitektur Komputer*. Sukoharjo : Tahta Media Group.

Utami, F. D., & Astuti, F. D. (2024). Comparison of Hadoop Mapreduce and Apache Spark in Big Data Processing with Hgrid247-DE. *JAIC : Journal of Applied Informatics and Computing*, *8*(2), 390–399. https://doi.org/10.30871/jaic.v8i2.8557

Werdiningsih, I., Novitasari, D. C. R., & Haq, D. Z. (2022). *Pengelolaan Data Mining dengan Pemrograman Matlab*. Surabaya : Airlangga University Press.

Zahwa, S., Amelia, N. D., Nafila, R., Putri, R. A., & Pujiono, I. P. (2025). Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++. *RESTIKOM : Riset Teknik Informatika Dan Komputer*, *7*(1), 123–136. https://doi.org/10.52005/restikom.v7i1.428