Evaluasi Kompleksitas Algoritma Sorting untuk Meningkatkan Efisiensi Pengolahan Data pada aplikasi

Muhammad Sulthan Fajri Rabbani [1], Muhammad Nasywan Amin [2], Abiansyah [3], Adib Fahmie Sudrajat [4]
[1,2,3,4]Universitas Islam Negeri Sultan Maulana Hasanuddin Banten, Indonesia
Email : wadplplk@gmail.com

Abstrak

Pertumbuhan aplikasi digital menyebabkan volume data yang diproses oleh sistem semakin besar dan beragam. Kondisi tersebut menuntut penggunaan algoritma yang efisien agar proses pencarian, pengurutan, penyajian, dan analisis data dapat berlangsung secara cepat. Penelitian ini bertujuan mengevaluasi kompleksitas beberapa algoritma sorting untuk meningkatkan efisiensi pengolahan data pada aplikasi digital. Algoritma yang dianalisis meliputi Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort. Penelitian ini menggunakan metode eksperimen komparatif melalui studi literatur, perancangan skenario pengujian, implementasi algoritma, serta analisis kompleksitas waktu dan ruang. Parameter evaluasi meliputi kompleksitas teoretis, kesesuaian algoritma terhadap karakteristik data, penggunaan memori, dan implikasinya terhadap performa aplikasi. Hasil evaluasi menunjukkan bahwa algoritma sederhana seperti Bubble Sort, Selection Sort, dan Insertion Sort cenderung memiliki kompleksitas O(n^2), sehingga kurang efisien ketika digunakan pada dataset berukuran besar. Sebaliknya, Merge Sort dan Quick Sort memiliki kompleksitas rata-rata O(n log n) dan lebih sesuai untuk aplikasi digital yang membutuhkan proses pengolahan data berskala menengah hingga besar. Temuan ini menegaskan bahwa pemilihan algoritma sorting berpengaruh langsung terhadap responsivitas sistem, efisiensi sumber daya komputasi, dan kualitas pengalaman pengguna. Dengan demikian, penelitian ini memberikan rekomendasi praktis bagi pengembang aplikasi dalam memilih algoritma sorting berdasarkan ukuran data, kebutuhan memori, dan karakteristik proses bisnis.
Kata kunci: algoritma sorting, kompleksitas algoritma, efisiensi data, aplikasi digital, pengolahan data

Abstract

The growth of digital applications has increased the volume and variety of data processed by information systems. This condition requires efficient algorithms so that searching, sorting, presenting, and analyzing data can be performed quickly. This study aims to evaluate the complexity of several sorting algorithms to improve data processing efficiency in digital applications. The analyzed algorithms include Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. This study employs a comparative experimental method through literature review, testing scenario design, algorithm implementation, and analysis of time and space complexity. The evaluation parameters include theoretical complexity, algorithm suitability for data characteristics, memory usage, and implications for application performance. The evaluation results indicate that simple algorithms such as Bubble Sort, Selection Sort, and Insertion Sort tend to have O(n^2) complexity, making them less efficient for large datasets. In contrast, Merge Sort and Quick Sort have an average complexity of O(n log n) and are more suitable for digital applications requiring medium to large-scale data processing. These findings confirm that selecting an appropriate sorting algorithm directly affects system responsiveness, computational resource efficiency, and user experience quality. Therefore, this study provides practical recommendations for application developers in selecting sorting algorithms based on data size, memory requirements, and business process characteristics.
Keywords: *sorting algorithm, algorithm complexity, data efficiency, digital application, data processing*

Pendahuluan

Perkembangan teknologi informasi mendorong peningkatan penggunaan aplikasi digital dalam berbagai bidang, seperti pendidikan, perdagangan elektronik, layanan kesehatan, sistem keuangan, dan administrasi publik. Aplikasi digital tidak hanya berfungsi sebagai media penyimpanan data, tetapi juga sebagai sistem yang memproses, menyaring, mengurutkan, dan menampilkan data secara cepat kepada pengguna. Kecepatan proses tersebut menjadi penting karena keterlambatan dalam pengolahan data dapat menurunkan responsivitas aplikasi dan mengganggu pengalaman pengguna.

Salah satu proses mendasar dalam pengolahan data adalah *sorting* atau pengurutan. Proses *sorting* digunakan untuk menyusun data berdasarkan kriteria tertentu, seperti nilai, waktu, nama, harga, prioritas, atau tingkat relevansi. Pada aplikasi *e-commerce*, *sorting* digunakan untuk mengurutkan produk berdasarkan harga atau popularitas. Pada sistem akademik, *sorting* digunakan untuk mengurutkan nilai mahasiswa, jadwal, atau data presensi. Pada sistem transaksi, *sorting* digunakan untuk menampilkan riwayat berdasarkan waktu terbaru atau jumlah transaksi. Dengan demikian, *sorting* merupakan operasi dasar yang memiliki peran penting dalam performa aplikasi digital.

Meskipun *sorting* terlihat sebagai proses sederhana, pemilihan algoritma yang kurang tepat dapat menimbulkan beban komputasi yang besar. Algoritma seperti Bubble Sort, Selection Sort, dan Insertion Sort mudah dipahami dan mudah diimplementasikan, tetapi memiliki keterbatasan ketika digunakan pada dataset besar. Sebaliknya, algoritma seperti Merge Sort dan Quick Sort memiliki strategi pemrosesan yang lebih efisien karena memanfaatkan pendekatan *divide and conquer*. Perbedaan tersebut menunjukkan bahwa evaluasi kompleksitas algoritma perlu dilakukan sebelum algoritma diterapkan pada sistem yang memproses data dalam jumlah besar.

Kompleksitas algoritma menjadi ukuran penting untuk menilai efisiensi suatu algoritma. Kompleksitas waktu menunjukkan hubungan antara ukuran input dengan waktu eksekusi yang dibutuhkan, sedangkan kompleksitas ruang menunjukkan besaran memori tambahan yang digunakan selama proses komputasi. Dalam pengembangan aplikasi, kedua aspek tersebut harus diperhatikan secara seimbang karena aplikasi yang cepat tetapi boros memori tetap berpotensi menimbulkan masalah pada perangkat dengan sumber daya terbatas.

Penelitian sebelumnya dalam bidang struktur data dan algoritma menunjukkan bahwa algoritma dengan kompleksitas O(n log n) umumnya lebih efisien dibandingkan algoritma O(n^2) ketika ukuran data meningkat secara signifikan (Cormen et al., 2022; Goodrich et al., 2021). Namun, pada praktik pengembangan aplikasi, pemilihan algoritma tidak hanya ditentukan oleh notasi kompleksitas, tetapi juga oleh karakteristik data, kebutuhan kestabilan urutan, penggunaan memori, dan konteks implementasi sistem. Oleh karena itu, evaluasi algoritma *sorting* perlu ditempatkan dalam konteks kebutuhan aplikasi digital, bukan hanya sebagai pembahasan teoretis.

Permasalahan utama yang diangkat dalam penelitian ini adalah bagaimana kompleksitas algoritma *sorting* memengaruhi efisiensi pengolahan data pada aplikasi digital. Penelitian ini juga mempertanyakan algoritma *sorting* mana yang lebih sesuai digunakan pada dataset berukuran kecil, sedang, dan besar. Tujuan penelitian ini adalah mengevaluasi kompleksitas Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort berdasarkan kompleksitas waktu, kompleksitas ruang, serta implikasinya terhadap performa aplikasi.

Novelty penelitian ini terletak pada penyajian evaluasi kompleksitas algoritma *sorting* yang dikaitkan secara langsung dengan kebutuhan efisiensi aplikasi digital. Berbeda dari pembahasan algoritma yang hanya menampilkan perbandingan teoretis, penelitian ini menempatkan hasil evaluasi sebagai dasar rekomendasi praktis bagi pengembang aplikasi dalam memilih algoritma *sorting* sesuai ukuran data, karakteristik input, serta keterbatasan sumber daya komputasi.

Metode

Penelitian ini menggunakan metode eksperimen komparatif dengan pendekatan analisis kompleksitas algoritma. Metode ini dipilih karena penelitian berfokus pada perbandingan karakteristik beberapa algoritma *sorting* berdasarkan kompleksitas waktu, kompleksitas ruang, dan kesesuaiannya

terhadap kebutuhan pengolahan data pada aplikasi digital. Algoritma yang dianalisis meliputi Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort.

Tahap pertama adalah studi literatur untuk mengidentifikasi konsep dasar, karakteristik, dan kompleksitas setiap algoritma. Literatur yang digunakan mencakup buku struktur data dan algoritma, rekayasa perangkat lunak, serta sumber akademik yang relevan dengan efisiensi komputasi. Studi literatur digunakan sebagai dasar penyusunan parameter evaluasi dan interpretasi hasil penelitian.

Tahap kedua adalah perancangan skenario pengujian. Skenario disusun berdasarkan variasi ukuran data dan karakteristik input. Ukuran data yang digunakan dalam rancangan evaluasi meliputi 1.000 data, 10.000 data, dan 50.000 data. Karakteristik input yang diperhatikan meliputi data acak, data hampir terurut, dan data terurut terbalik. Variasi ini digunakan untuk menggambarkan kondisi yang sering muncul pada aplikasi digital.

Tahap ketiga adalah implementasi algoritma dalam bentuk prototipe pengujian. Setiap algoritma dirancang dengan prinsip modular agar proses perbandingan dapat dilakukan secara konsisten. Prototipe pengujian bertujuan mengamati kecenderungan performa algoritma ketika ukuran data meningkat. Implementasi juga memperhatikan kesamaan lingkungan pengujian agar setiap algoritma memperoleh beban kerja yang sebanding.

Tahap keempat adalah analisis hasil evaluasi. Analisis dilakukan dengan membandingkan kompleksitas teoretis dan implikasi praktis setiap algoritma. Kompleksitas waktu dikaji berdasarkan *best case*, *average case*, dan *worst case*, sedangkan kompleksitas ruang dikaji berdasarkan kebutuhan memori tambahan selama proses pengurutan. Hasil evaluasi kemudian dihubungkan dengan konteks aplikasi digital, seperti sistem database, aplikasi *web*, aplikasi *mobile*, dan sistem rekomendasi.

Tabel 1. Algoritma yang Dievaluasi dalam Penelitian

No.	Algoritma	Pendekatan Utama	Karakteristik Umum
1	Bubble Sort	Pertukaran berulang	Sederhana, tetapi tidak efisien untuk data besar
2	Selection Sort	Pemilihan nilai minimum/maksimum	Jumlah pertukaran relatif sedikit, tetapi perbandingan tetap tinggi
3	Insertion Sort	Penyisipan elemen	Efisien untuk data kecil atau hampir terurut
4	Merge Sort	Divide and conquer	Stabil dan efisien untuk data besar, tetapi membutuhkan memori tambahan
5	Quick Sort	Divide and conquer	Cepat pada kondisi rata-rata, tetapi sensitif terhadap pemilihan pivot

Berdasarkan Tabel 1, penelitian ini mengevaluasi lima algoritma *sorting* yang memiliki pendekatan dan karakteristik berbeda. *Bubble Sort* menggunakan pendekatan pertukaran berulang sehingga mudah dipahami, tetapi kurang efisien untuk pengolahan data berukuran besar. *Selection Sort* bekerja dengan memilih nilai minimum atau maksimum pada setiap iterasi, sehingga jumlah pertukarannya relatif sedikit, meskipun jumlah perbandingannya tetap tinggi. *Insertion Sort* menerapkan mekanisme penyisipan elemen dan cenderung efisien ketika digunakan pada data kecil atau data yang hampir terurut. Sementara itu, *Merge Sort* dan *Quick Sort* menggunakan pendekatan *divide and conquer*. *Merge Sort* dikenal stabil dan efisien untuk data besar, tetapi membutuhkan memori tambahan, sedangkan *Quick Sort* memiliki performa cepat pada kondisi rata-rata, meskipun kinerjanya dapat dipengaruhi oleh pemilihan *pivot*.

Tabel 2. Rancangan Skenario Evaluasi Data

No.	Ukuran Data	Karakteristik Input	Tujuan Evaluasi
1	1.000 data	Acak, hampir terurut, terbalik	Menilai efisiensi pada dataset kecil
2	10.000 data	Acak, hampir terurut, terbalik	Menilai perubahan performa pada dataset sedang
3	50.000 data	Acak, hampir terurut, terbalik	Menilai skalabilitas algoritma pada dataset besar

Parameter evaluasi dalam penelitian ini meliputi waktu eksekusi, kompleksitas algoritma, penggunaan memori, *stabilitas* pengurutan, dan kesesuaian implementasi pada aplikasi digital. Waktu eksekusi digunakan untuk menggambarkan kecepatan proses pengurutan. Kompleksitas algoritma digunakan untuk menilai pertumbuhan beban komputasi terhadap peningkatan jumlah data. Penggunaan

memori digunakan untuk melihat kebutuhan ruang tambahan. Stabilitas pengurutan digunakan untuk menilai kemampuan algoritma mempertahankan urutan relatif elemen yang memiliki nilai sama. Kesesuaian implementasi digunakan untuk menentukan algoritma yang paling relevan pada konteks aplikasi tertentu.

Hasil dan Pembahasan

Hasil Evaluasi Kompleksitas Teoretis

Evaluasi kompleksitas menunjukkan bahwa setiap algoritma *sorting* memiliki karakteristik yang berbeda. Bubble Sort, Selection Sort, dan Insertion Sort termasuk algoritma sederhana yang memiliki kompleksitas waktu rata-rata dan terburuk sebesar O(n^2). Kompleksitas tersebut menunjukkan bahwa peningkatan jumlah data akan menyebabkan pertumbuhan jumlah operasi yang sangat besar. Dengan demikian, ketiga algoritma ini kurang sesuai untuk dataset berukuran besar pada aplikasi digital yang menuntut respons cepat.

Merge Sort dan Quick Sort menunjukkan karakteristik yang lebih efisien. Merge Sort memiliki kompleksitas waktu O(n log n) pada kondisi terbaik, rata-rata, dan terburuk. Keunggulan ini membuat Merge Sort stabil dan dapat diprediksi, khususnya untuk data dalam jumlah besar. Quick Sort memiliki kompleksitas rata-rata O(n log n), tetapi dapat menurun menjadi O(n^2) pada kondisi terburuk apabila pemilihan pivot tidak optimal. Meskipun demikian, Quick Sort tetap banyak digunakan karena performa rata-ratanya cepat dan penggunaan memorinya relatif lebih rendah dibandingkan Merge Sort.

Tabel 3. Perbandingan Kompleksitas Algoritma Sorting

Algoritma	Best Case	Average Case	Worst Case	Space Complexity	Stabilitas
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)	Stabil
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)	Tidak stabil
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)	Stabil
Merge Sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Stabil
Quick Sort	O(n log n)	O(n log n)	O(n^2)	O(log n)	Tidak stabil

Analisis Efisiensi terhadap Ukuran Data

Pada dataset kecil, perbedaan performa antaralgoritma tidak selalu terlihat secara signifikan. Insertion Sort bahkan dapat menjadi pilihan yang baik ketika data berukuran kecil atau hampir terurut karena proses penyisipannya relatif sederhana. Kondisi ini menunjukkan bahwa algoritma sederhana tidak selalu buruk, selama digunakan pada konteks yang sesuai. Namun, ketika ukuran data meningkat, kompleksitas O(n^2) menjadi kelemahan utama karena jumlah operasi bertambah secara kuadratik.

Pada dataset sedang dan besar, algoritma dengan kompleksitas O(n log n) lebih layak digunakan. Merge Sort memberikan performa stabil karena proses pembagian dan penggabungan data berlangsung konsisten. Quick Sort cenderung unggul pada banyak kasus praktis karena pemrosesan dilakukan secara efisien dengan kebutuhan memori tambahan yang lebih rendah. Namun, Quick Sort memerlukan strategi pemilihan pivot yang baik agar tidak mengalami penurunan performa pada kondisi terburuk.

Tabel 4. Rekomendasi Penggunaan Algoritma *Berdasarkan* Karakteristik Data

Karakteristik Data	Algoritma yang Disarankan	Alasan
Data kecil	Insertion Sort	Implementasi sederhana dan efisien untuk data hampir terurut
Data hampir terurut	Insertion Sort	Jumlah perpindahan data relatif sedikit
Data besar dan stabilitas penting	Merge Sort	Kompleksitas konsisten dan stabil
Data besar dengan memori terbatas	Quick Sort	Rata-rata cepat dan kebutuhan memori tambahan lebih rendah
Aplikasi pembelajaran algoritma	Bubble Sort atau Selection Sort	Mudah digunakan untuk memahami konsep dasar perbandingan dan pertukaran

Implikasi pada Aplikasi Digital

Dalam aplikasi digital, pemilihan algoritma *sorting* tidak hanya berdampak pada kecepatan proses, tetapi juga pada kualitas layanan yang diterima pengguna. Aplikasi *e-commerce*, misalnya, memerlukan pengurutan produk berdasarkan harga, ulasan, popularitas, atau waktu pembaruan. Apabila algoritma yang digunakan tidak efisien, proses pemuatan daftar produk dapat menjadi lambat, terutama ketika jumlah data meningkat. Pada kondisi tersebut, algoritma seperti Merge Sort atau Quick Sort lebih relevan dibandingkan Bubble Sort atau Selection Sort.

Pada sistem manajemen database dan *dashboard* administrasi, *sorting* sering digunakan untuk mengurutkan transaksi, laporan, presensi, atau data pengguna. Proses pengurutan yang lambat dapat memengaruhi kecepatan sistem dalam menghasilkan laporan. Penggunaan algoritma dengan kompleksitas lebih rendah membantu sistem menjaga responsivitas, terutama ketika data terus bertambah seiring waktu.

Pada aplikasi *mobile*, efisiensi algoritma perlu mempertimbangkan keterbatasan CPU, RAM, dan konsumsi daya. Algoritma yang boros operasi dapat menyebabkan aplikasi terasa lambat atau tidak responsif. Oleh karena itu, pengembang perlu memilih algoritma yang sesuai dengan kapasitas perangkat dan ukuran data yang dikelola aplikasi. Quick Sort dapat menjadi pilihan untuk kebutuhan cepat dengan memori relatif efisien, sedangkan Merge Sort lebih sesuai ketika stabilitas urutan menjadi kebutuhan utama.

Tabel 5. Dampak Pemilihan Algoritma Sorting pada Aplikasi Digital

Aspek Aplikasi	Dampak Algoritma Tidak Efisien	Solusi yang Disarankan
Responsivitas antarmuka	Daftar data lambat ditampilkan	Gunakan algoritma O(n log n) untuk data besar
Kinerja server	Beban CPU meningkat	Optimalkan pemilihan algoritma dan struktur data
Penggunaan memori	Aplikasi mudah melambat pada perangkat terbatas	Pilih algoritma dengan space complexity rendah
Pengalaman pengguna	Pengguna menunggu lebih lama	Minimalkan latency proses pengurutan
Skalabilitas sistem	Performa turun saat data bertambah	Gunakan algoritma yang skalabel

Pembahasan

Hasil evaluasi menunjukkan bahwa kompleksitas algoritma memiliki hubungan langsung dengan efisiensi pengolahan data. Algoritma yang memiliki kompleksitas O(n^2) masih dapat digunakan untuk kebutuhan sederhana, tetapi tidak direkomendasikan untuk aplikasi yang memproses data besar. Hal ini disebabkan oleh pertumbuhan jumlah operasi yang tidak sebanding dengan peningkatan kebutuhan responsivitas sistem.

Bubble Sort merupakan algoritma yang mudah dipahami karena bekerja dengan membandingkan dan menukar elemen bersebelahan secara berulang. Namun, kemudahan implementasi tersebut tidak sejalan dengan efisiensi pada data besar. Selection Sort memiliki jumlah pertukaran yang lebih sedikit dibandingkan Bubble Sort, tetapi jumlah perbandingannya tetap tinggi. Insertion Sort lebih adaptif terhadap data yang hampir terurut, sehingga masih relevan pada kondisi tertentu.

Merge Sort dan Quick Sort memberikan efisiensi yang lebih baik karena menggunakan pendekatan *divide and conquer*. Merge Sort unggul pada stabilitas dan konsistensi kompleksitas, sedangkan Quick Sort unggul pada performa rata-rata dan efisiensi memori. Perbedaan ini menunjukkan bahwa tidak ada satu algoritma yang paling baik untuk semua kondisi. Algoritma terbaik bergantung pada ukuran data, distribusi data, kebutuhan stabilitas, dan sumber daya sistem.

Dalam konteks pengembangan aplikasi, hasil penelitian ini menegaskan pentingnya analisis algoritmik pada tahap perancangan sistem. Pengembang tidak cukup hanya memastikan bahwa fitur *sorting* berjalan, tetapi juga perlu memastikan bahwa fitur tersebut tetap efisien ketika jumlah data meningkat. Kesalahan pemilihan algoritma pada tahap awal dapat menyebabkan penurunan performa ketika aplikasi digunakan oleh lebih banyak pengguna atau ketika data bertambah secara signifikan.

Dengan demikian, evaluasi kompleksitas algoritma *sorting* dapat menjadi dasar pengambilan keputusan teknis dalam pengembangan aplikasi digital. Untuk kebutuhan data kecil, Insertion Sort masih

dapat digunakan karena sederhana dan adaptif. Untuk kebutuhan data besar, Merge Sort atau Quick Sort lebih disarankan. Apabila stabilitas data menjadi prioritas, Merge Sort lebih tepat. Apabila efisiensi memori dan kecepatan rata-rata menjadi prioritas, Quick Sort dapat menjadi pilihan yang lebih sesuai.

Kesimpulan

Berdasarkan hasil evaluasi, dapat disimpulkan bahwa pemilihan algoritma *sorting* berpengaruh signifikan terhadap efisiensi pengolahan data pada aplikasi digital. Algoritma sederhana seperti Bubble Sort, Selection Sort, dan Insertion Sort memiliki kelebihan dari sisi kemudahan implementasi, tetapi kurang efisien untuk dataset besar karena kompleksitas rata-rata dan terburuknya berada pada O(n^2). Algoritma tersebut lebih sesuai digunakan untuk pembelajaran konsep dasar atau pengolahan data kecil.

Merge Sort dan Quick Sort menunjukkan efisiensi yang lebih baik untuk kebutuhan aplikasi digital berskala menengah hingga besar karena memiliki kompleksitas rata-rata O(n log n). Merge Sort lebih unggul ketika stabilitas urutan data dibutuhkan, sedangkan Quick Sort lebih sesuai ketika pengembang membutuhkan performa rata-rata cepat dengan penggunaan memori tambahan yang relatif rendah. Dengan demikian, pemilihan algoritma harus mempertimbangkan ukuran data, karakteristik input, kebutuhan stabilitas, dan keterbatasan sumber daya komputasi.

Penelitian ini merekomendasikan agar pengembang aplikasi melakukan evaluasi kompleksitas sebelum menentukan algoritma *sorting* yang digunakan dalam sistem. Penelitian selanjutnya dapat memperluas evaluasi dengan pengujian empiris menggunakan dataset *real-time*, menambahkan algoritma lain seperti Heap Sort, Tim Sort, dan Radix Sort, serta mengukur performa pada berbagai bahasa pemrograman dan perangkat keras.

Daftar Pustaka

Ansori, A., Damyati, F., & Dhestyani, S. A. (2025). Assessing AI Integration in Islamic Higher Education: A Mixed-Methods Fishbone Diagram Analysis. IJID (International Journal on Informatics for Development), 13(2), 504–516.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2022). Introduction to algorithms (4th ed.). MIT Press.

Goodrich, M. T., Tamassia, R., & Goldwasser, M. H. (2021). Data structures and algorithms in Python. Wiley.

Knuth, D. E. (1998). The art of computer programming, volume 3: Sorting and searching (2nd ed.). Addison-Wesley.

Lafore, R. (2017). Data structures and algorithms in Java (2nd ed.). Sams Publishing.

Pressman, R. S., & Maxim, B. R. (2020). Software engineering: A practitioner's approach (9th ed.). McGraw-Hill.

Rokhmah, S., Permana, D., Elmi, F., & Ansori, A. (2025). AI implementation as support for successful green campus implementation. Educational Process: International Journal, 19, e2025569.

Sedgewick, R., & Wayne, K. (2011). Algorithms (4th ed.). Addison-Wesley.

Shaffer, C. A. (2013). Data structures and algorithm analysis. Dover Publications.

Skiena, S. S. (2020). The algorithm design manual (3rd ed.). Springer.

Weiss, M. A. (2014). Data structures and algorithm analysis in Java (3rd ed.). Pearson.

Wirth, N. (2004). Algorithms and data structures. Prentice Hall.