TAMIKA: Jurnal Tugas Akhir Manajemen Informatika & Komputerisasi Akuntansi
Vol. 5 No. 2 (Desember 2025)
https://doi.org/10.46880/tamika.Vol5No2.pp344-349 e-ISSN: 2828-1276

PERBANDINGAN KINERJA OPERATOR == DAN FUNGSI COMPARE() DALAM PEMROSESAN STRING PADA C++

Muhammad Rofi Nur Fasyih✉, Dea Karuniawati, Pasha Ferbitama, Elvina Rizqi, Imam Prayogo Pujiono
Fakultas Ekonomi dan Bisnis Islam, UIN K.H. Abdurrahman Wahid, Pekalongan, Indonesia
Email: muhammad.rofi.nur25007@mhs.uingusdur.ac.id

ABSTRACT

This study compared the performance of the equality operator (==) and the compare() function in C++ for strings. The focus was on their speed and memory efficiency when processing large data sets. Three datasets containing 100, 1,000, and 10,000 random strings ranging in length from 5 to 10 characters were used, each tested three times. Time was measured in microseconds, while memory usage was calculated in kilobytes. The results showed that the compare() function was faster, about 42% faster on the 100-string dataset, and about 9% to 10% faster on larger datasets. Memory usage was similar, differing by only a few hundred bytes. Therefore, use the equality operator (==) for simple comparisons or on small datasets, and use the compare() function for larger data sets. Further research should examine the impact of different compilers, optimization levels, and C++ standard library implementations on these results, as well as include formal statistical analysis of the measured data.

Keywords: String Comparison, Performance Analysis, C++, Compare Function, Comparison Operators.

ABSTRAK

Penelitian ini membandingkan cara kerja operator kesetaraan (==) dan fungsi compare() dalam C++ untuk string. Fokusnya adalah pada seberapa cepat dan efisien penggunaan memori saat memproses data besar. Dalam eksperimen ini, digunakan tiga dataset yang berisi 100, 1.000, dan 10.000 string acak dengan panjang antara 5 sampai 10 karakter, dan setiap dataset diuji sebanyak tiga kali. Waktu yang dibutuhkan diukur dalam mikrodetik, sedangkan penggunaan memori dihitung dalam kilobyte. Hasilnya menunjukkan bahwa fungsi compare() lebih cepat yaitu sekitar 42% lebih baik pada dataset dengan 100 string, dan sekitar 9% hingga 10% lebih cepat pada dataset yang lebih besar. Untuk penggunaan memori, hasilnya hampir sama, hanya berbeda beberapa ratus byte. Maka dari itu, gunakan operator kesetaraan (==) untuk perbandingan yang mudah atau pada dataset yang kecil, dan gunakan fungsi compare() untuk data yang besar. Penelitian lanjutan perlu mengkaji pengaruh variasi kompiler, tingkat optimasi, dan implementasi pustaka standar C++ terhadap hasil ini, serta menambahkan analisis statistik formal terhadap data pengukuran.

Kata Kunci: Perbandingan String, Analisis Kinerja, C++, Fungsi Compare, Operator Perbandingan.

PENDAHULUAN

Efisiensi komputasi sangat penting dalam pengembangan perangkat lunak modern karena secara langsung berdampak pada kinerja, pengalaman pengguna, skalabilitas, dan biaya operasional (Rustiyana et al., 2025). Di era aplikasi yang semakin kompleks, efisiensi memastikan perangkat lunak berjalan dengan cepat, responsif, dan mengoptimalkan penggunaan sumber daya komputer (Aji et al., 2024). C++ dikenal karena performa dan efisiensi memorinya yang tinggi berkat beberapa fitur utama, terutama karena bahasa ini memberikan kontrol tingkat rendah langsung atas memori dan perangkat keras kepada programmer (Zahwa et al., 2025). Dalam bahasa C++, kita akan menemukan pilihan antara dua metode yang dapat dilakukan untuk menyimpan data tekstual seperti nama, alamat, pesan, dan kode lainnya. Dua metode yang dimaksud adalah Operator Perbandingan == dan Fungsi Compare(). Dalam penerapannya, kedua metode itu memiliki persamaan dalam tugas, bahkan bisa dikatakan sama persis. Namun, dalam suatu kondisi tertentu, kita mungkin akan dihadapkan dengan opsi untuk memilih menggunakan Operator Perbandingan == atau Fungsi Compare(). Maka dari itu, penelitian ini akan membahas perbedaan antara Operasi Perbandingan == dan Fungsi Compare() pada bahasa pemrograman C++ berdasarkan waktu eksekusi dan penggunaan memori. Penelitian ini diharapkan dapat memberikan manfaat teoritis berupa kontribusi terhadap kajian efisiensi algoritmik dalam pemrograman C++, serta manfaat praktis berupa rekomendasi bagi pengembang untuk memilih metode perbandingan string yang paling efisien sesuai konteks penggunaan. Adapun batasan penelitian ini meliputi pengujian hanya pada string dengan operator == dan fungsi compare() tanpa

melibatkan fungsi C-style seperti strcmp, serta pembatasan variabel pengujian pada waktu eksekusi dan penggunaan memori di lingkungan kompilasi tertentu (Kusnadi & Wicaksono, 2017).

Penelitian ini bertujuan untuk membandingkan kinerja dua metode perbandingan string dalam bahasa C++, yaitu operator bawaan == dan fungsi compare(). Pengujian dilakukan terhadap tiga ukuran dataset yang terdiri dari 100, 1.000, dan 10.000 string acak. Setiap string memiliki panjang antara 5 hingga 10 karakter dan dibangkitkan secara acak menggunakan generator bilangan acak Mersenne Twister (Siswanto, n.d.). Dua parameter utama diukur untuk menilai kinerja kedua metode ini, yakni waktu eksekusi dalam satuan mikrodetik dan perkiraan penggunaan memori dalam satuan kilobyte (Anwar & Hayadi, 2020).

TINJAUAN PUSTAKA

Konsep Dasar Bahasa Pemrograman C++

Dalam bahasa pemrograman C++ terdapat konsep-konsep yang menjadi dasar dari mempelajari bahasa pemrograman C++ tersebut. Diantara konsep dasar tersebut ada yang namanya algoritma, yaitu instruksi yang ditetapkan untuk menyelesaikan masalah atau program tertentu, dan juga pseudocode, yaitu representasi kode program dalam bahasa manusia yang lebih mudah dipahami, serta struktur dari program C++ itu sendiri yang mencakup deklarasi, fungsi utama (main) dan statement (Guntara & Hapsan, 2023).

Sejarah Singkat dan Karakteristik C++

C++ merupakan bahasa pemrograman yang ditemukan pada tahun 1983 oleh Bjarne Stroustrup di Bell Labs (Trianiza et al., 2025). Beberapa fitur utama C++ adalah kemampuan untuk mengorganisir kode dalam objek, kemampuan untuk menggunakan pointer, penggunaan kelas untuk membuat objek, dan kemampuan untuk melakukan overloading operator. Selain itu C++ juga mendukung konsep pemrograman generik, yaitu kemampuan untuk menulis kode yang dapat digunakan untuk membuat berbaagai jenis data.Salah satu faktor yang membuat bahasa pemrograman populer yaitu karena mudah untuk digunakan, kemudahan bahasa pemrograman dapat mempengaruhi cepat atau tidaknya aplikasi karena peroses pengkodeannya sangat cepat. Selain itu, bahasa pemrograman C++ ini juga dapat menyederhanakan perintah yang kompleks menjadi perintah yang lebih sederhana karena bahasa pemrograman ini berisi perintah yang sangat panjang dan bahasa yang sulit untuk dipahami, sehingga membutuhkan banyak waktu bagi komputer untuk menerjemahkannya (Trianiza et al., 2025).

Konsep Object-Oriented Programming (OOP) dalam C++

Berdasarkan penelitian yang dilakukan oleh (Retnoningsih et al., 2017), Pemrograman berorientasi objek atau object oriented programming (OOP) merupakan suatu pendekatan pemrograman yang menggunakan object dan class. OOP memberikan kemudahan dalam pembuatan sebuah program, keuntungan yang didapat apabila membuat Program berorientasi objek atau object oriented programming (OOP) antara lain:

1) Reusability, kode yang dibuat dapat digunakan kembali
2) Extensibility, pemrogram dapat membuat methode baru atau mengubah yang sudah ada sesuai yang diinginkan tanpa harus membuat kode dari awal
3) Maintainability, kode yang sudah dibuat lebih mudah untuk dikelola apabila aplikasi yang dibuat berskala besar yang memungkinkan adanya error dalam pengembangannya hal tersebut dapat diatasi dengan OOP.

Konsep Dasar String dalam C++

String adalah sebuah representasi data dalam komputer yang berupa kumpulan dari karakter/ huruf yang disimpan dalam sebuah array/ tabel yang memiliki indeks. Dengan demikian, string dapat dengan mudah diakses karakter-karakternya (dengan mengakses indeks tabel) dan membandingkannya dengan string lain. Inti dari pengolahan string adalah Penambahan (Insertion), Penghapusan (Deletion), Penukaran (Substitution), dan Pencocokan String (Matching). Penambahan adalah jika suatu string ditambahkan dengan string lain atau jika ke dalam suatu string ditambahkan suatu karakter. Penghapusan adalah jika di dalam suatu string, dihilangkan satu karakter atau lebih. Penukaran adalah jika suatu karakter dalam suatu string diganti dengan karakter lain yang berbeda. Pencocokan String adalah bagaimana suatu string dibandingkan dengan string lain dan dilihat kesamaan karakter-karakternya (Ardiyanto, 2008).

Perbandingan String dalam C++

Perbandingan Operator dan Fungsi compare() dalam C++. Dalam C++, objek std::string bisa dibandingkan menggunakan operator relasional seperti ==, <, >, dan lainnya, atau dengan fungsi anggota compare() (Rianna & Siahaan, 2020).

Dengan diperkenalkannya std::string_view pada C++17, efisiensi operasi pembacaan string meningkat karena objek ini tidak menyalin data string, hanya menunjuk ke buffer yang sudah ada (Kovacs et al., 2024).

Operator == lebih disarankan digunakan untuk memeriksa kesamaan sederhana karena lebih singkat dan sedikit lebih cepat dalam beberapa skenario(Dinata & Hasdyna, 2025).

Fungsi compare() lebih cocok digunakan untuk perbandingan terstruktur atau berdasarkan urutan abjad, atau saat membandingkan bagian tertentu dari string (Syamsuri et al., 2024)

Peran Pustaka Standar (Standard Template Library – STL) dalam Pengolahan Data.

Standar Template Library (STL) berperan dalam menyediakan struktur data dan algoritma umum agar menghemat waktu (Putra, 2024) .

Pengertian Operator

Operator dalam bahasa C++ adalah simbol yang dipergunakan untuk mengerjakan atau memanipulasi suatu operasi, selain itu operator biasanya digunakan untuk perkalian dua nilai, pemberian nilai pada variabel dan perbandingan antara dua nilai (Anam et al., 2021).

METODE PENELITIAN

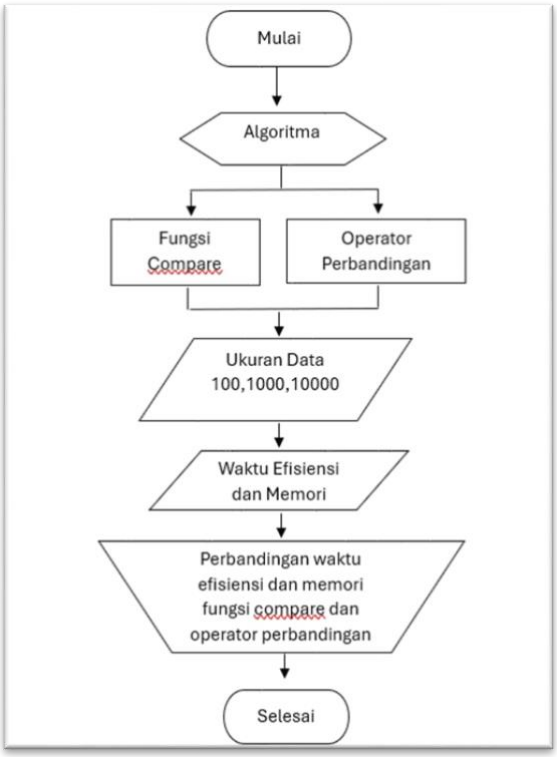
Penelitian ini dilaksanakan dengan menggunakan bahasa pemrograman C++ untuk mengeksekusi sebuah kode. Pengujian dilakukan pada laptop dengan spesifikasi sebagai berikut:

- Prosesor : 1x 11th Generation Intel® Core™ i3-1115G4 Processor(Core™ i3-1115G4)
- SSD : 512 GB
- Memori RAM : 8 GB
- Sistem operasi : Windows 11 Home Single Language 64(EN:English)

Penelitian ini menggunakan pendekatan kuantitatif dengan metode eksperimen komparatif untuk membandingkan efisiensi kinerja antara operator perbandingan dan fungsi compare() dalam memproses string. Metode eksperimen dipilih karena memungkinkan pengujian langsung terhadap kinerja algoritma, sehingga menghasilkan data yang bisa diukur. Penelitian dilakukan untuk mengevaluasi secara sistematis kinerja algoritma antara operator perbandingan dan fungsi compare() dalam pengolahan string.

Untuk memperjelas proses, penelitian mencakup beberapa tahapan seperti persiapan dataset, penerapan algoritma, pengujian, pengumpulan data, dan analisis perbandingan. Tahap eksperimen dimulai dengan pembuatan dataset yang dibagi menjadi tiga kategori berdasarkan ukurannya, yaitu ukuran kecil (100 elemen), ukuran sedang, (1.000 elemen) dan ukuran besar (10.000 elemen) untuk menguji kemampuan skala algoritma. Selanjutnya, algoritma operator perbandingan dan fungsi compare() dalam pengolahan string diimplementasikan menggunakan bahasa pemrograman C++, dan kode program ditulis serta dijalankan melalui Code::Blocks. Pengujian kinerja dilakukan dengan menjalankan setiap algoritma pada kedua ukuran dataset tersebut untuk mengukur efisiensi proses pengolahan string.

Uji coba dilakukan secara berulang pada setiap dataset agar hasilnya konsisten. Data yang dikumpulkan berupa efisiensi kinerja selama proses berlangsung. Setelah itu, data hasil pengujian dianalisis dan dibandingkan untuk menentukan algoritma mana yang memiliki kinerja terbaik. Hasil pengujian ini diharapkan bisa memberikan pemahaman yang jelas mengenai performa nyata kedua algoritma string dalam skenario dataset berskala sedang hingga besar.



Gambar 1. Ilustrasi proses penelitian

HASIL DAN PEMBAHASAN

Eksperimen dilakukan untuk membandingkan kinerja dua metode perbandingan string dalam bahasa C++, yaitu:

1. Operator perbandingan bawaan (==) dan,
2. Fungsi string::compare()

Kedua metode diuji dengan tiga ukuran dataset, yaitu 100, 1.000, dan 10.000 string acak. Setiap string dibuat secara acak dengan panjang antara 5–10 karakter. Kinerja yang diukur meliputi dua aspek, meliputi waktu eksekusi (microdetik), yaitu lama proses perbandingan antar string (Maulana, 2016) dan perkiraan penggunaan memori (byte), yaitu alokasi

memori dari dua vector<string> yang menampung dataset dan seluruh karakter di dalamnya.

Eksperimen ini dilakukan dengan membandingkan dua metode perbandingan string dalam bahasa C++: pertama, operator perbandingan bawaan (==), dan kedua, fungsi compare(). Pengujian dilakukan pada tiga ukuran dataset: 100 string, 1.000 string, dan 10.000 string masing-masing string dibangkitkan secara acak dengan panjang antara 5–10 karakter. Aspek kinerja yang diukur meliputi (1) waktu eksekusi (mikrodetik) untuk membandingkan antar-string, dan (2) perkiraan penggunaan memori (kilobyte) dari dua vector<string> yang menampung dataset beserta seluruh karakter.

Tabel 1. Hasil Pengujian Waktu Eksekusi Untuk Kedua Metode Perbandingan String dengan 3 Kali Percobaan

Ukuran Data	Percobaan Ke	Operator Perbandingan (==) (µs)	Fungsi compare() (µs)
100	1	3	5
	2	12	2
	3	4	4
Rata-rata		6,33	3,67
1.000	1	24	30
	2	26	18
	3	35	29
Rata-rata		28,33	25,67
10.000	1	198	262
	2	286	223
	3	282	210
Rata-rata		255,33	231,67

Dari hasil tersebut, dapat dilihat bahwa pada dataset paling kecil (100 string), fungsi compare() menunjukkan waktu yang lebih cepat dibanding operator ==. Namun seiring meningkatnya ukuran dataset hingga 1.000 dan 10.000 string, selisih kinerja antara kedua metode semakin mengecil, dan metode compare() tetap menunjukkan sedikit keunggulan waktu.

Tabel 2. Hasil Pengujian Perkiraan Pemakaian Memori Untuk Kedua Metode Perbandingan String dengan 3 Kali Percobaan

Ukuran Data	Percobaan Ke	Operator Perbandingan (==) (KB)	Fungsi compare() (KB)
100	1	7,79	7,79
	2	7,75	7,24
	3	7,76	7,77
Rata-rata		7,78	7,43
1.000	1	77,05	77,31
	2	77,21	77,27
	3	77,15	77,26
Rata-rata		77,14	77,28
10.000	1	771,39	771,26
	2	771,62	771,85
	3	771,52	771,69
Rata-rata		771,51	771,6

Dari hasil tersebut menunjukkan bahwa penggunaan memori antara kedua metode hampir identik, dengan selisih sangat kecil yang secara praktik tidak signifikan. Nilai yang sedikit lebih tinggi untuk metode compare() pada dataset kecil kemungkinan terjadi akibat overhead tambahan dari pemanggilan fungsi dan nilai kembalian integer.

Berdasarkan hasil eksperimen yang ditampilkan pada Tabel 1.1, diketahui bahwa pada ukuran data 100 string, rata-rata waktu eksekusi untuk operator == adalah 6,33 µs, sedangkan untuk fungsi compare() adalah 3,67 µs. Pada ukuran data 1.000 string, operator == mencatat waktu rata-rata 28,33 µs, sedangkan fungsi compare() sedikit lebih cepat dengan 25,67 µs. Kemudian, pada ukuran data terbesar yaitu 10.000 string, operator == membutuhkan waktu rata-rata 255,33 µs, sedangkan fungsi compare() menunjukkan hasil 231,67 µs. Data tersebut menunjukkan bahwa pada dataset kecil, perbedaan waktu antara kedua metode masih cukup terasa, namun semakin kecil seiring meningkatnya ukuran dataset. Menariknya, meskipun operator == tampak lebih sederhana karena

hanya membandingkan nilai boolean, hasil percobaan justru memperlihatkan bahwa fungsi compare() sedikit lebih efisien secara waktu, terutama pada dataset besar. Hal ini menunjukkan adanya optimalisasi internal pada implementasi compare() dalam pustaka standar C++, di mana proses pembandingan dapat dilakukan secara lebih terstruktur dengan efisiensi manajemen karakter di tingkat memori.

Sementara itu, hasil pengukuran penggunaan memori pada Tabel 1.2 memperlihatkan bahwa perbedaan antara kedua metode hampir tidak signifikan. Untuk ukuran data 100 string, rata-rata penggunaan memori oleh operator == sebesar 7,78 KB, sedangkan fungsi compare() sebesar 7,43 KB. Pada dataset 1.000 string, penggunaan memori oleh operator == mencapai 77,14 KB dan fungsi compare() sebesar 77,28 KB. Adapun pada dataset 10.000 string, hasilnya meningkat menjadi 771,51 KB untuk operator == dan 771,60 KB untuk fungsi compare(). Dari nilai-nilai tersebut terlihat bahwa perbedaan hanya berkisar pada beberapa ratus byte, yang dalam konteks komputasi modern tergolong sangat kecil dan tidak berpengaruh signifikan terhadap kinerja keseluruhan. Sedikit peningkatan pada fungsi compare() kemungkinan disebabkan oleh adanya nilai kembalian bertipe integer dan overhead pemanggilan fungsi, yang menambah sedikit beban pada stack memory selama proses eksekusi (Zahwa et al., 2025).

Secara umum, hasil penelitian ini menunjukkan bahwa fungsi compare() memiliki performa yang lebih stabil dibandingkan operator == ketika jumlah data meningkat. Pada skala kecil, operator == sering kali lebih disukai karena sintaksnya lebih sederhana dan mudah dibaca. Namun, ketika jumlah string yang dibandingkan semakin besar, efisiensi fungsi compare() menjadi lebih terlihat. Performa operasi string lintas bahasa pemrograman juga menunjukkan bahwa perbedaan kecil dalam implementasi fungsi string dapat memengaruhi efisiensi total program, terutama ketika jumlah operasi string meningkat secara eksponensial.

Secara keseluruhan, penelitian ini dapat disimpulkan bahwa fungsi compare() memberikan kinerja waktu eksekusi yang sedikit lebih baik dan stabil dibanding operator == dalam skenario pengolahan string berukuran besar, tanpa perbedaan signifikan dalam konsumsi memori. Hasil ini mengindikasikan bahwa pemilihan metode perbandingan string di C++ dapat mempertimbangkan kompleksitas data dan kebutuhan aplikasi. Untuk proses perbandingan yang sederhana dan cepat, operator == tetap menjadi pilihan ideal karena lebih ringkas. Namun, pada aplikasi yang memerlukan pembandingan string berskala besar atau berulang, penggunaan fungsi compare() lebih direkomendasikan karena konsistensi performanya.

KESIMPULAN

Berdasarkan hasil penelitian yang telah dilakukan, dalam bahasa pemrograman C++ terdapat metode perbandingan string yang bisa dilakukan dengan menggunakan dua cara pengoperasian, yaitu menggunakan operator perbandingan == dan compare(). Yang mana dapat disimpulkan bahwa keduanya memiliki kinerja yang relatif serupa dari segi penggunaan memori, akan tetapi menunjukkan perbedaan dalam aspek waktu eksekusinya. Fungsi compare() cenderung memberikan performa yang lebih stabil dan efisien apabila jumlah data yang dibandingkan meningkat. Sementara itu, operator == menunjukkan kecepatan yang baik pada dataset kecil dikarena proses perbandingannya yang lebih langsung tanpa pemanggilan fungsi tambahan. Pada eksperimen dengan dataset hingga 10.000 string, fungsi compare() menghasilkan waktu eksekusi rata-rata lebih rendah dibandingkan operator ==, dengan perbedaan waktu efisiensi sekitar ± 9–10%. Hasil ini mengindikasikan bahwa pemilihan metode perbandingan string sebaiknya disesuaikan dengan skala dan kompleksitas program, operator == lebih sesuai untuk operasi sederhana dan cepat, sedangkan fungsi compare() direkomendasikan untuk pengolahan data besar yang memerlukan konsistensi kinerja. Penelitian lanjutan sebaiknya diarahkan untuk menguji kinerja kedua metode pada dataset yang berukuran lebih besar, panjang string yang bervariasi, serta pengaruh perbedaan implementasi *Standard Template Library* (*STL*) di berbagai compiler C++, sehingga diperoleh pemahaman yang lebih komprehensif mengenai efisiensi komputasi dalam pemrosesan string.

DAFTAR PUSTAKA

Aji, B. R. K., Herulambang, W., & Setyatama, F. (2024). Perancangan dan Implementasi Aplikasi Enterprise Resource Planning (ERP) untuk Meningkatkan Efisiensi Bisnis (Studi Kasus CV. OTW Computer Gusaha). *Prosiding TAU SNARS-TEK Seminar Nasional Rekayasa Dan Teknologi*, *3*(1), 69–75.

Anam, S., Yanti, I., Fitriah, Z., & Habibah, U. (2021). *Cara Mudah Belajar Bahasa Pemrograman C++*. Universitas Brawijaya Press.

Anwar, H., & Hayadi, B. H. (2020). *Set Instruksi Dan Kinerja: Mengoptimalkan Arsitektur Komputer Untuk Aplikasi Modern*.

Ardiyanto, R. I. (2008). Dynamic programming dalam levenshtein distance untuk mengetahui keterbedaan dua string. *Skripsi. Institut Teknologi Bandung*.

Dinata, R. K., & Hasdyna, N. (2025). *Algoritma dan Pemrograman: Konsep Dasar, Logika, dan Implementasi dengan C++ & Python*. Serasi Media Teknologi.

Guntara, R. G., & Hapsan, A. (2023). *Algoritma dan Pemrograman Dasar: Menggunakan Bahasa Pemrograman C++ dengan Contoh Kasus Aplikasi untuk Bisnis dan Manajemen*. CV. Ruang Tentor.

Kovacs, R., Horvath, G., & Porkolab, Z. (2024). Detecting lifetime errors of std:: string_view objects in C++. *ArXiv Preprint ArXiv:2408.09325*.

Kusnadi, A., & Wicaksono, A. K. (2017). Perbandingan Algoritma Horspool dan Algoritma Zhu-Takaoka dalam Pencarian String Berbasis Desktop. *Ultima Computing: Jurnal Sistem Komputer*, *9*(1), 12–16.

Maulana, R. (2016). Analisa perbandingan kompleksitas algoritma selectionsort dan insertionsort. *Jurnal Informatika*, *3*(2).

Putra, R. A. (2024). *Buku Sakti Pemrograman C++ Untuk Pemula*. Anak Hebat Indonesia.

Retnoningsih, E., Shadiq, J., & Oscar, D. (2017). Pembelajaran Pemrograman Berorientasi Objek (Object Oriented Programming) Berbasis Project Based Learning. *Informatics For Educators And Professional: Journal of Informatics*, *2*(1), 95-â.

Rianna, M., & Siahaan, L. F. (2020). *Buku Ajar Bahasa Pemrograman C++*. GUEPEDIA.

Rustiyana, R., Judijanto, L., Datya, A. I., Lase, M. C. M., Hidayat, M. M., Wardani, A. T., Rijal, M., Prayudani, S., Cahyono, D., & Bahana, R. (2025). *Pengantar Teknologi Informasi*. PT. Sonpedia Publishing Indonesia.

Siswanto, L. A. (2016). *Analisis Keacakan Generator Angka Pseudorandom Mersenne Twister dengan Metode Diehard Test*.

Syamsuri, A. R., Afriyah, A., Septiawati, A., & Tunisa, N. M. (2024). *Aplikasi Komputer Statistik Dengan SPSS*. Merdeka Kreasi Group.

Trianiza, I., Khirdany, E. N., Wahyudi, E., Vandika, A. Y., & Sofyan, S. (2025). Pengenalan Pemrograman Dasar Dunia Koding dengan C++. *YPAD Penerbit*.

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++. *Jurnal RESTIKOM: Riset Teknik Informatika Dan Komputer*, *7*(1), 123–136.