

EVALUASI EMPIRIS KINERJA DAN STABILITAS BUBBLE SORT, SELECTION SORT, DAN INSERTION SORT PADA PENGURUTAN DATA MAHASISWA BERBASIS C++

Fakhri Yuhans Nurfawwas, Muhamad Ishom El Saha,
Neysa Alzena, Raditya Satya Wardhana, Imam Prayogo Pujiono
Program Studi Informatika, Fakultas Sains dan Teknologi,
Universitas Islam Negeri K.H. Abdurrahman Wahid Pekalongan, Indonesia
fakhri.yuhans.nurfawwas25080@mhs.uingusdur.ac.id

ABSTRAK

Pengurutan data mahasiswa dibutuhkan untuk menata daftar hadir, rekap akademik, dan arsip administrasi agar pencarian data lebih mudah dilakukan. Permasalahan penelitian ini terletak pada perlunya evaluasi empiris terhadap *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* pada *record* mahasiswa berbasis C++ dengan kunci NIM dan nama. Penelitian ini bertujuan membandingkan ketiga algoritma berdasarkan waktu eksekusi dan stabilitas hasil pengurutan. Metode yang digunakan berupa eksperimen komputasional menggunakan C++17, struktur data *vector<Mahasiswa>*, *dataset* sintetis berukuran 100, 1.000, dan 5.000 *record*, serta pengukuran waktu dengan *std::chrono* pada tiga skenario pengujian. Hasil pengujian memperlihatkan bahwa *Selection Sort* mencatat waktu eksekusi paling rendah pada seluruh skenario, *Insertion Sort* berada di posisi menengah, sedangkan *Bubble Sort* membutuhkan waktu paling tinggi. Pada aspek stabilitas, *Bubble Sort* dan *Insertion Sort* mempertahankan urutan relatif *record* dengan kunci sama, sementara *Selection Sort* standar tidak stabil. Berdasarkan hasil tersebut, *Selection Sort* lebih tepat untuk kebutuhan yang menekankan waktu eksekusi pada konfigurasi pengujian ini, sedangkan *Insertion Sort* lebih relevan ketika stabilitas urutan *record* dengan kunci yang sama menjadi prioritas.

Kata kunci: pengurutan data, data mahasiswa, NIM, *Bubble Sort*, *Selection Sort*, *Insertion Sort*

1. PENDAHULUAN

Pengurutan merupakan operasi dasar dalam komputasi yang digunakan untuk menata data ke dalam urutan tertentu agar proses pencarian, pemeriksaan, pengelompokan, dan pelaporan dapat dilakukan secara lebih terstruktur. Dalam pengelolaan data akademik, pengurutan diperlukan pada daftar mahasiswa, daftar hadir, rekap kelas, arsip administrasi, dan tampilan layanan akademik. NIM digunakan sebagai identitas yang konsisten, sedangkan nama lebih mudah dikenali ketika data ditampilkan atau dicetak. Oleh karena itu, pengurutan berdasarkan NIM dan nama memiliki peran praktis dalam pengelolaan data mahasiswa [2], [3], [8].

Permasalahan utama dalam penelitian ini adalah pemilihan algoritma pengurutan dasar yang sesuai ketika data mahasiswa diuji berdasarkan waktu eksekusi dan stabilitas hasil. *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* masih sering digunakan sebagai algoritma dasar dalam pembelajaran dan implementasi sederhana, tetapi hasil empirisnya dapat berbeda karena dipengaruhi bahasa pemrograman, struktur data, pola data masukan, serta cara implementasi [3], [4]. Beberapa penelitian C++ juga lebih banyak membahas data numerik atau algoritma yang lebih modern, sehingga pengujian pada *record* mahasiswa dengan kunci NIM dan nama masih perlu diperjelas [5], [6], [7], [9].

Penelitian ini bertujuan mengimplementasikan *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* pada data mahasiswa berbentuk *record* serta membandingkan kinerjanya berdasarkan waktu

eksekusi dan stabilitas. Penyelesaian masalah dilakukan melalui eksperimen komputasional menggunakan C++17, data mahasiswa sintetis, kunci pengurutan NIM dan nama, struktur data *vector<Mahasiswa>*, pengukuran waktu menggunakan *std::chrono*, serta validasi stabilitas pada data nama dengan duplikasi tinggi. Melalui rancangan tersebut, penelitian ini diarahkan untuk memberi dasar evaluasi sederhana dalam memilih algoritma pengurutan pada aplikasi akademik berskala kecil hingga menengah.

2. TINJAUAN PUSTAKA

Konsep *sorting* merujuk pada proses menyusun sekumpulan data berdasarkan aturan urutan tertentu. Pada implementasi praktis, algoritma pengurutan dipilih bukan hanya karena notasi kompleksitasnya, tetapi juga karena sifat stabilitas dan perilakunya terhadap pola data masukan [1], [11].

Bubble Sort bekerja dengan membandingkan elemen yang berdekatan, lalu menukar posisinya apabila urutannya salah. Algoritma ini sederhana, mudah divisualisasikan, dan umum dipakai sebagai materi dasar pembelajaran, tetapi cenderung lambat ketika ukuran data membesar [3], [8], [11].

Selection Sort memilih elemen minimum dari bagian data yang belum terurut, kemudian menempatkannya pada posisi yang semestinya. Jumlah pertukaran pada algoritma ini relatif lebih sedikit dibanding *Bubble Sort*, tetapi implementasi standarnya tidak stabil dan tetap berada pada kelas waktu kuadratik [1], [2]. Dalam konteks aplikasi

akademik, *Selection Sort* juga pernah digunakan untuk pengurutan keaktifan siswa berdasarkan data kehadiran [2].

Insertion Sort membangun bagian data yang sudah terurut sedikit demi sedikit dengan cara menyisipkan elemen baru ke posisi yang tepat. Algoritma ini dikenal baik untuk data kecil atau data

yang tidak jauh dari kondisi terurut, serta tetap stabil dalam implementasi standarnya [3], [9], [12].

Stabilitas algoritma berarti urutan relatif elemen yang memiliki kunci sama tetap dipertahankan pada hasil akhir. Sifat ini penting pada data mahasiswa, khususnya ketika pengurutan dilakukan berdasarkan nama, karena dua atau lebih mahasiswa dapat memiliki nama yang sama [1].

Tabel 1. Karakter umum algoritma yang digunakan

Algoritma	Cara kerja ringkas	Kekuatan utama	Kelemahan utama
<i>Bubble Sort</i>	Membandingkan elemen berdekatan lalu menukarnya jika salah urut	Mudah dipahami dan divisualisasikan	Paling lambat saat data membesar
<i>Selection Sort</i>	Memilih elemen minimum dari bagian yang belum terurut	Jumlah pertukaran relatif sedikit	Implementasi standar tidak stabil dan tetap kuadratik
<i>Insertion Sort</i>	Menyisipkan elemen ke bagian yang sudah terurut	Efisien untuk data kecil hingga menengah dan sederhana diimplementasikan	Kinerja tetap menurun saat ukuran data makin besar

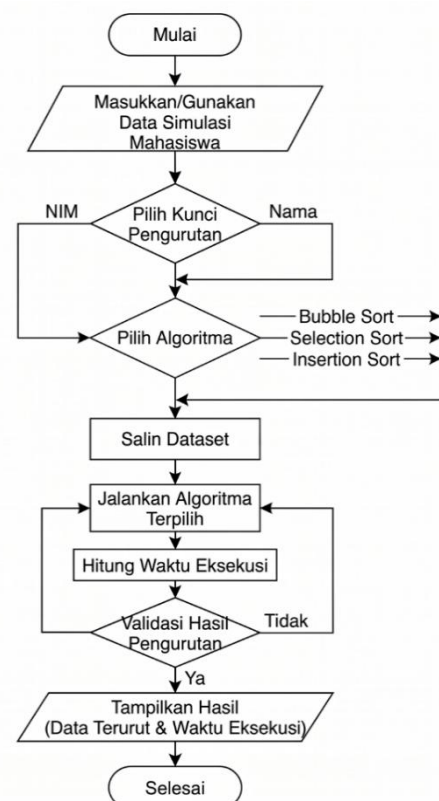
Tabel 1 menunjukkan bahwa ketiga algoritma termasuk algoritma pengurutan dasar dengan karakter operasional yang berbeda. Secara umum, *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* memiliki kecenderungan waktu kuadratik pada kasus rata-rata atau terburuk. Namun, perbedaan cara kerja, jumlah pertukaran, dan sifat stabilitas membuat hasil empirisnya dapat berbeda ketika diterapkan pada *record* mahasiswa [1], [3], [8], [11], [12].

Penelitian sebelumnya memperlihatkan pola yang cukup konsisten. Pada data mahasiswa dan data numerik, *Insertion Sort* sering menunjukkan hasil lebih baik daripada *Bubble Sort* pada skala kecil hingga menengah, sedangkan *Bubble Sort* tetap bernilai sebagai alat bantu pembelajaran karena langkahnya mudah ditelusuri [3], [9], [10]. Di sisi lain, studi pada implementasi C++ dan perbandingan lintas pendekatan menunjukkan bahwa performa empiris juga dipengaruhi oleh bahasa pemrograman, struktur data, dan strategi implementasi yang digunakan [4], [5], [6], [7], [9]. Posisi penelitian ini berada pada ruang yang lebih sempit namun relevan, yaitu pengurutan *record* mahasiswa dengan fokus pada NIM, nama, dan stabilitas hasil.

3. METODE PENELITIAN

3.1. Desain Penelitian

Penelitian ini menggunakan pendekatan eksperimen komputasional pada program konsol berbasis C++. Fokus penelitian terletak pada implementasi dan analisis kinerja tiga algoritma pengurutan dasar, yaitu *Bubble Sort*, *Selection Sort*, dan *Insertion Sort*, ketika diterapkan pada data mahasiswa berbentuk *record*. Rancangan pengujian dibuat agar setiap algoritma dijalankan pada *dataset* yang sama untuk menjaga konsistensi perbandingan, mengikuti pola eksperimen komparatif pada penelitian sebelumnya [3], [5], [6], [7], [9].



Gambar 1. Flowchart alur penelitian pengurutan data mahasiswa

Gambar 1 menggambarkan alur penelitian secara ringkas. Data mahasiswa terlebih dahulu dimasukkan atau dibangkitkan, kemudian pengguna menentukan kunci pengurutan dan algoritma yang akan dipakai. Dataset asli disalin sebelum algoritma dijalankan agar setiap metode bekerja pada kondisi awal yang sama. Setelah proses pengurutan selesai, waktu eksekusi diukur, hasilnya divalidasi, lalu ditampilkan sebagai dasar analisis.

3.2. Struktur Data Mahasiswa

Data mahasiswa direpresentasikan sebagai *record* dengan dua atribut utama, yaitu NIM dan nama. Dalam implementasi C++17, data disimpan menggunakan struktur Mahasiswa dan dikelola melalui *vector<Mahasiswa>*. Pemilihan *vector* dilakukan karena struktur ini mudah diakses secara indeks, mudah disalin untuk pengujian berulang, dan sesuai untuk implementasi algoritma pengurutan dasar pada data berukuran kecil hingga menengah. Pemilihan atribut NIM dan nama dilakukan karena keduanya memiliki fungsi yang berbeda dalam pengelolaan data akademik. NIM digunakan sebagai identitas yang lebih terstruktur dan cenderung unik, sedangkan nama digunakan sebagai atribut yang lebih mudah dikenali oleh pengguna. Dengan menggunakan dua kunci tersebut, pengujian dapat memperlihatkan bagaimana algoritma bekerja pada atribut identitas dan atribut tekstual. Hal ini penting karena program pengurutan data mahasiswa tidak hanya dituntut menghasilkan urutan yang benar, tetapi juga tetap mudah digunakan dalam konteks administrasi sederhana.

3.3. Implementasi Algoritma

Ketiga algoritma diimplementasikan secara manual pada *record* mahasiswa, bukan pada *array* bilangan bulat. Proses perbandingan dilakukan berdasarkan atribut yang dipilih pengguna. Jika pengguna memilih NIM, maka perbandingan menggunakan nilai NIM. Jika pengguna memilih nama, maka perbandingan menggunakan nilai nama. *Bubble Sort* diimplementasikan dengan pertukaran berulang antarelemen yang berdekatan, *Selection Sort* dengan pencarian elemen minimum pada bagian yang belum terurut, dan *Insertion Sort* dengan penyisipan elemen ke posisi yang tepat pada bagian data yang telah terurut [2], [8], [12].

3.4. Skenario Pengujian

Pengujian dilakukan pada *dataset* sintesis berukuran 100, 1.000, dan 5.000 *record*. Dataset dibangkitkan menggunakan generator *pseudo-random* pada program C++. NIM disusun sebagai *string* numerik 10 digit agar menyerupai identitas mahasiswa, sedangkan nama dipilih dari kumpulan nama yang telah disiapkan. Tiga skenario digunakan dalam penelitian ini. Skenario pertama adalah pengurutan berdasarkan NIM pada data acak. Skenario kedua adalah pengurutan berdasarkan nama pada data acak. Skenario ketiga adalah pengurutan berdasarkan nama pada data dengan duplikasi tinggi. Pada skenario duplikasi tinggi, variasi nama dibuat lebih terbatas sehingga beberapa *record* memiliki nama yang sama. Skenario ini dipilih karena nama mahasiswa dalam praktik administrasi tidak selalu unik, sehingga stabilitas algoritma menjadi aspek yang relevan untuk diuji [1], [3].

3.5. Pengukuran dan Validasi

Pengukuran waktu eksekusi dilakukan untuk memastikan bahwa perbandingan antaralgoritma berjalan pada kondisi yang sama. Waktu yang diukur hanya mencakup proses pengurutan, sehingga tidak mencakup pembangkitan data, penyalinan *dataset*, maupun pencetakan keluaran. Setiap skenario dijalankan sebanyak sepuluh kali, kemudian hasilnya dirata-ratakan untuk mengurangi kemungkinan bias dari satu kali eksekusi.

Tabel 2. Perangkat keras dan aplikasi yang digunakan

Komponen	Keterangan
Spesifikasi perangkat keras	Laptop Lenovo V14 G2 ALC, AMD Ryzen 3 5300U, RAM 12 GB DDR4 3200 MHz, SSD 256 GB NVMe Gen 3
Bahasa pemrograman	C++17
Editor kode	Visual Studio Code
Compiler	g++ 14.2.0
Lingkungan <i>build</i>	MSYS2 MinGW64
Sistem operasi	Windows 10 LTSC
Pengukur waktu	<i>std::chrono</i>
Struktur data	<i>vector<Mahasiswa></i>

Tabel 2 menunjukkan lingkungan implementasi dan pengujian program yang digunakan dalam penelitian ini. Validasi dilakukan dalam dua tahap. Tahap pertama memastikan bahwa hasil akhir benar-benar terurut sesuai kunci yang dipilih, baik berdasarkan NIM maupun nama. Tahap kedua dilakukan pada pengurutan nama dengan duplikasi tinggi untuk menguji stabilitas algoritma. Uji stabilitas penting karena pada data mahasiswa terdapat kemungkinan beberapa *record* memiliki nama yang sama. Dalam kondisi tersebut, algoritma yang stabil akan mempertahankan urutan relatif data sesuai posisi awalnya, sedangkan algoritma yang tidak stabil dapat mengubah urutan *record* dengan kunci yang sama.

4. HASIL DAN PEMBAHASAN

Program yang dibuat dapat mengurutkan data mahasiswa berdasarkan dua kunci, yaitu NIM dan nama. Hasil validasi menunjukkan bahwa ketiga algoritma menghasilkan urutan yang sesuai dengan kunci yang dipilih. Perbedaan utama muncul pada waktu eksekusi dan stabilitas masing-masing algoritma ketika dijalankan pada skenario pengujian yang sama.

Tabel 3. Waktu rata-rata pengurutan berdasarkan NIM

Ukuran data	<i>Bubble Sort</i> (ms)	<i>Selection Sort</i> (ms)	<i>Insertion Sort</i> (ms)
100	1,187	0,196	0,331
1.000	112,648	15,819	30,880
5.000	2.726,471	380,555	810,459

Berdasarkan Tabel 3, pengurutan berdasarkan NIM menunjukkan pola yang konsisten. *Selection Sort* mencatat waktu paling rendah pada seluruh ukuran data, *Insertion Sort* berada di posisi kedua, dan *Bubble Sort* membutuhkan waktu paling tinggi. Pada ukuran

100 *record*, selisih waktu masih kecil secara absolut. Namun, setelah ukuran data meningkat menjadi 1.000 dan 5.000 *record*, perbedaan waktu antaralgoritma menjadi lebih jelas. Pola ini memperlihatkan bahwa meskipun ketiga algoritma sama-sama berada pada kelompok pengurutan dasar dengan kecenderungan waktu kuadratik, cara kerja setiap algoritma tetap memengaruhi waktu eksekusi ketika diterapkan pada *record* mahasiswa [1], [3], [8], [12].

Pada ukuran data 5.000 *record* berdasarkan NIM, *Selection Sort* mencatat waktu 380,555 ms. Pada ukuran yang sama, *Insertion Sort* membutuhkan 810,459 ms, sedangkan *Bubble Sort* mencapai 2.726,471 ms. Selisih tersebut membuat *Selection Sort* sekitar 2,13 kali lebih cepat dibandingkan *Insertion Sort* dan sekitar 7,16 kali lebih cepat dibandingkan *Bubble Sort*. Perbedaan ini berkaitan dengan jumlah pertukaran data pada *Selection Sort* yang relatif lebih sedikit dibandingkan dua algoritma lainnya pada konfigurasi pengujian ini.

Tabel 4. Waktu rata-rata pengurutan berdasarkan nama pada data acak

Ukuran data	<i>Bubble Sort</i> (ms)	<i>Selection Sort</i> (ms)	<i>Insertion Sort</i> (ms)
100	1,100	0,217	0,350
1.000	106,461	17,832	30,727
5.000	2.755,118	427,903	803,268

Tabel 4 memperlihatkan bahwa perubahan kunci pembanding dari NIM ke nama tidak mengubah urutan kinerja algoritma pada data acak. *Selection Sort* tetap berada pada posisi tercepat, *Insertion Sort* berada di posisi menengah, dan *Bubble Sort* menjadi yang paling lambat. Pada data mahasiswa skala kecil hingga menengah, pembandingan *string* pada atribut nama masih menghasilkan pola kinerja yang sejalan dengan pengurutan berdasarkan NIM. Hal ini memperkuat bahwa performa empiris tidak hanya ditentukan oleh kompleksitas teoretis, tetapi juga dipengaruhi oleh karakter implementasi dan lingkungan pengujian [4], [5].

Pada pengurutan nama acak ukuran 5.000 *record*, waktu eksekusi *Bubble Sort* adalah 2.755,118 ms, *Selection Sort* 427,903 ms, dan *Insertion Sort* 803,268 ms. Angka tersebut menegaskan bahwa *Selection Sort* tetap mencatat waktu paling rendah pada skenario ini. Dibandingkan *Insertion Sort*, *Selection Sort* berjalan sekitar 1,88 kali lebih cepat. Jika dibandingkan *Bubble Sort*, selisihnya lebih besar, yaitu sekitar 6,44 kali lebih cepat. Dengan demikian, pada pengujian nama acak, *Selection Sort* tetap menjadi algoritma dengan waktu eksekusi paling rendah.

Tabel 5. Waktu rata-rata pengurutan berdasarkan nama pada data duplikasi tinggi

Ukuran data	<i>Bubble Sort</i> (ms)	<i>Selection Sort</i> (ms)	<i>Insertion Sort</i> (ms)
100	1,040	0,208	0,326
1.000	93,205	18,641	31,561
5.000	2.325,547	448,798	645,763

Tabel 5 menunjukkan bahwa data dengan duplikasi nama tinggi tidak mengubah urutan performa algoritma. *Selection Sort* tetap mencatat waktu eksekusi paling rendah, *Insertion Sort* berada di posisi kedua, dan *Bubble Sort* membutuhkan waktu paling tinggi. Skenario ini penting karena pada data mahasiswa, nama yang sama dapat muncul lebih dari sekali. Oleh karena itu, pengujian tidak cukup hanya melihat kecepatan, tetapi juga perlu memperhatikan apakah urutan relatif *record* dengan nama yang sama tetap terjaga.

Pada data duplikasi tinggi ukuran 5.000 *record*, *Bubble Sort* membutuhkan waktu 2.325,547 ms, *Selection Sort* 448,798 ms, dan *Insertion Sort* 645,763 ms. Waktu *Bubble Sort* dan *Insertion Sort* pada skenario ini lebih rendah dibandingkan pengurutan nama acak, tetapi posisi *Selection Sort* sebagai algoritma dengan waktu eksekusi terendah tetap tidak berubah. Dengan demikian, duplikasi nama memengaruhi nilai waktu pada beberapa algoritma, tetapi tidak mengubah kesimpulan utama dari pengujian.

Perbedaan hasil penelitian ini dengan beberapa penelitian terdahulu, terutama yang menempatkan *Insertion Sort* lebih baik pada kondisi tertentu, dapat dipengaruhi oleh bentuk data yang digunakan. Penelitian ini memakai *record* mahasiswa yang memuat *string*, implementasi manual berbasis C++17, serta lingkungan *compiler* dan perangkat yang spesifik. Karena itu, hasil penelitian ini sebaiknya dibaca sebagai temuan empiris pada konfigurasi pengujian tertentu, bukan sebagai klaim universal bahwa *Selection Sort* selalu lebih baik daripada *Insertion Sort* pada semua jenis data [3], [4], [5].

Tabel 6. Hasil uji stabilitas pada pengurutan nama dengan duplikasi tinggi

Algoritma	Hasil stabilitas
<i>Bubble Sort</i>	Stabil pada implementasi standar
<i>Selection Sort</i>	Tidak stabil pada implementasi standar
<i>Insertion Sort</i>	Stabil pada implementasi standar

Tabel 6 memperlihatkan perbedaan stabilitas ketiga algoritma pada pengurutan nama dengan duplikasi tinggi. *Bubble Sort* dan *Insertion Sort* bersifat stabil pada implementasi standar karena tetap mempertahankan urutan relatif *record* dengan kunci yang sama. Sebaliknya, *Selection Sort* standar tidak stabil karena proses pemilihan elemen minimum dan pertukaran posisi dapat mengubah urutan relatif data yang memiliki nama sama. Sifat ini sesuai dengan penjelasan teoretis dalam literatur [1], [2], [12].

Stabilitas menjadi pertimbangan penting ketika pengurutan dilakukan berdasarkan nama. Dalam data mahasiswa, dua atau lebih *record* dapat memiliki nama yang sama. Jika sistem hanya memprioritaskan waktu eksekusi pada konfigurasi pengujian ini, *Selection Sort* dapat dipilih karena mencatat waktu paling rendah. Namun, apabila aplikasi perlu mempertahankan urutan

relatif *record* dengan nama yang sama, *Insertion Sort* lebih sesuai dibandingkan *Selection Sort* karena bersifat stabil dan tetap lebih cepat daripada *Bubble Sort*.

Bubble Sort menjadi algoritma dengan waktu eksekusi paling tinggi karena melakukan pertukaran berulang pada elemen yang berdekatan. Ketika ukuran data bertambah, jumlah operasi perpindahan data ikut meningkat sehingga waktu eksekusinya menjadi lebih besar. Pola ini sejalan dengan penelitian sebelumnya yang menempatkan *Bubble Sort* sebagai algoritma dasar dengan efisiensi waktu rendah pada data acak atau data yang mulai membesar [3], [8], [11].

Selection Sort mencatat waktu eksekusi paling rendah pada seluruh skenario pengujian. Keunggulan ini dapat dikaitkan dengan jumlah pertukaran yang lebih terbatas dibandingkan *Bubble Sort* dan *Insertion Sort*. Walaupun *Selection Sort* tetap menelusuri bagian data yang belum terurut untuk mencari elemen minimum, jumlah perpindahan data yang lebih sedikit membuat waktu eksekusinya lebih rendah pada program dan perangkat yang digunakan. Kelemahan utamanya tetap berada pada aspek stabilitas, karena implementasi standar *Selection Sort* tidak mempertahankan urutan relatif *record* dengan nama identik [2], [3], [8].

Insertion Sort berada pada posisi menengah dari sisi waktu eksekusi. Algoritma ini tidak secepat *Selection Sort* pada hasil pengujian, tetapi secara konsisten lebih cepat daripada *Bubble Sort*. Nilai penting *Insertion Sort* dalam penelitian ini terletak pada sifat stabilnya. Oleh karena itu, *Insertion Sort* dapat menjadi pilihan yang lebih seimbang ketika data memiliki nama yang sama dan aplikasi perlu menjaga urutan relatif *record* tersebut [3], [9], [12].

Jika dibandingkan dengan penelitian C++ lain, hasil penelitian ini perlu ditempatkan secara proporsional. Studi komparatif pada delapan algoritma *sorting* [5], *QR Sort* [6], *Intro Sort* [9], serta pendekatan rekursif dan iteratif [7] menunjukkan bahwa algoritma yang lebih adaptif atau lebih modern dapat memberi hasil lebih baik pada data numerik yang lebih besar. Penelitian ini tidak diarahkan untuk menggantikan temuan tersebut, melainkan memperlihatkan perilaku tiga algoritma dasar ketika diterapkan pada *record* mahasiswa dengan skenario implementasi sederhana.

Secara keseluruhan, ukuran data berpengaruh terhadap jarak waktu eksekusi antaralgoritma. Pada ukuran 100 *record*, perbedaannya masih kecil sehingga ketiga algoritma masih dapat digunakan untuk kebutuhan sederhana. Pada ukuran 1.000 dan 5.000 *record*, perbedaan waktu menjadi lebih nyata. Kondisi ini menunjukkan bahwa pemilihan algoritma tetap perlu diperhatikan, meskipun data yang digunakan belum termasuk kategori sangat besar.

Dari sisi penggunaan praktis, rekomendasi algoritma bergantung pada kebutuhan aplikasi. *Selection Sort* layak dipilih ketika prioritas utama adalah waktu eksekusi rendah pada konfigurasi

program serupa. Namun, pilihan tersebut memiliki konsekuensi karena implementasi standarnya tidak stabil. *Bubble Sort* dan *Insertion Sort* sama-sama stabil, tetapi *Bubble Sort* membutuhkan waktu paling tinggi pada seluruh skenario. Dengan pertimbangan tersebut, *Insertion Sort* lebih sesuai ketika stabilitas data nama duplikat diperlukan. Untuk kebutuhan pembelajaran, ketiga algoritma tetap layak digunakan karena alurnya mudah dipahami dan dapat membantu menjelaskan konsep pengurutan dasar [10]. Batasan penelitian ini adalah *dataset* yang masih bersifat sintesis, ukuran data yang dibatasi sampai 5.000 *record*, serta pengujian yang berfokus pada waktu eksekusi dan stabilitas, sementara penggunaan memori belum dianalisis secara rinci [5].

5. KESIMPULAN DAN SARAN

Penelitian ini menunjukkan bahwa *Bubble Sort*, *Selection Sort*, dan *Insertion Sort* dapat digunakan untuk mengurutkan *record* mahasiswa berdasarkan NIM dan nama, tetapi ketiganya menghasilkan kinerja yang berbeda pada *dataset* 100, 1.000, dan 5.000 *record*. Berdasarkan hasil pengujian pada Laptop Lenovo V14 G2 ALC dengan prosesor AMD Ryzen 3 5300U, RAM 12 GB, penyimpanan 256 GB NVMe SSD, sistem operasi Windows 10 LTSC, IDE Visual Studio Code, *compiler* g++ 14.2.0, dan lingkungan MSYS2 MinGW64, *Selection Sort* mencatat waktu eksekusi paling rendah pada seluruh skenario pengujian. Pada ukuran 5.000 *record*, *Selection Sort* memperoleh waktu 380,555 ms untuk pengurutan berdasarkan NIM, 427,903 ms untuk pengurutan nama pada data acak, dan 448,798 ms untuk pengurutan nama pada data duplikasi tinggi. *Insertion Sort* berada pada posisi menengah dengan waktu 810,459 ms pada pengurutan NIM, 803,268 ms pada nama acak, dan 645,763 ms pada nama duplikasi tinggi. *Bubble Sort* menjadi algoritma dengan waktu eksekusi paling tinggi, yaitu 2.726,471 ms pada pengurutan NIM, 2.755,118 ms pada nama acak, dan 2.325,547 ms pada nama duplikasi tinggi. Dari sisi stabilitas, *Bubble Sort* dan *Insertion Sort* mampu mempertahankan urutan relatif *record* dengan nama yang sama, sedangkan *Selection Sort* standar tidak stabil. Oleh karena itu, *Selection Sort* lebih sesuai digunakan apabila prioritas utama adalah waktu eksekusi pada konfigurasi pengujian serupa, sedangkan *Insertion Sort* lebih relevan apabila pengurutan berdasarkan nama memerlukan stabilitas urutan *record*. Untuk penelitian selanjutnya, penggunaan data mahasiswa riil, perluasan ukuran *dataset*, analisis penggunaan memori, dan penambahan pembandingan dari pustaka standar maupun algoritma yang lebih modern dapat dilakukan agar evaluasi kinerja menjadi lebih menyeluruh.

DAFTAR PUSTAKA

- [1] S. Sundaramoorthy and G. Karunanidhi, "A systematic analysis on performance and computational complexity of sorting

- algorithms,” *Discover Computing*, vol. 28, no. 1, pp. 1-30, 2025, doi: 10.1007/s10791-025-09724-w.
- [2] H. Pramudya, A. F. Cobantoro, and J. Karaman, “Implementasi Algoritma Selection Sort dalam Sistem Absensi Siswa untuk Pengurutan Keaktifan Berdasarkan Kehadiran: Studi Kasus SMA Negeri 1 Nawangan Pacitan,” *JATI (Jurnal Mahasiswa Teknik Informatika)*, vol. 9, no. 2, pp. 2887-2895, 2025, doi: 10.36040/jati.v9i2.13212.
- [3] G. Gustina, A. H. Ahadi, F. H. Aminuddin, and M. F. Syawal, “Performance Analysis of Bubble Sort, Selection Sort, and Insertion Sort Algorithms Using Python on Student Data,” *JIPTI: Jurnal Inovasi Pendidikan dan Teknologi Informasi*, vol. 6, no. 2, pp. 529-536, 2025, doi: 10.52060/jipti.v6i2.3792.
- [4] J. Iskandar, H. Suhendar, and B. D. Pamungkas, “Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python,” *G-Tech: Jurnal Teknologi Terapan*, vol. 8, no. 1, pp. 104-113, 2024, doi: 10.33379/gtech.v8i1.3556.
- [5] M. I. Ali, R. D. Fardiarsyah, L. Shodik, F. Z. D. Kinanti, and I. P. Pujiono, “Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++,” *LogicLink*, vol. 2, no. 1, pp. 1-17, 2025, doi: 10.28918/logiclink.v2i1.10868.
- [6] A. Handika, H. Azzahrani, R. Karima, M. Mualim, and I. P. Pujiono, “Analisis Efisiensi Memori dan Waktu Eksekusi Algoritma QR Sort dan Bubble Sort dalam C++,” *JEIS: Jurnal Elektro dan Informatika Swadharma*, vol. 6, no. 1, pp. 118-129, 2026, doi: 10.56486/jeis.vol6no1.1060.
- [7] S. Zahwa, N. D. Amelia, R. Nafila, R. A. Putri, and I. P. Pujiono, “Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++,” *Jurnal RESTIKOM: Riset Teknik Informatika dan Komputer*, vol. 7, no. 1, pp. 123-136, 2025, doi: 10.52005/restikom.v7i1.428.
- [8] M. F. Abdullah, I. Hafiza, R. Wahyuni, and A. Syahputra, “Penggunaan Algoritma Bubble Sort dalam Pengurutan Nomor Induk Mahasiswa,” *Hello World Jurnal Ilmu Komputer*, vol. 2, no. 1, pp. 14-19, 2023, doi: 10.56211/helloworld.v2i1.206.
- [9] S. N. Fatmaluna, N. Gista, A. Rahma, S. Aulia, and I. P. Pujiono, “Comparison of Memory Efficiency and Computation Time of Bubble Sort, Insertion Sort, and Intro Sort Algorithms Using the C++ Programming Language,” *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, vol. 5, no. 2, pp. 2308-2319, 2026, doi: 10.59934/jaiea.v5i2.1858.
- [10] P. D. Setyorini, L. T. Azzahro, R. A. Yuniar, and I. P. Pujiono, “Pengembangan Alat Bantu Pembelajaran Sorting Algorithm Berbasis Visual Console C++,” *Jurnal Riset dan Aplikasi Mahasiswa Informatika (JRAMI)*, vol. 6, no. 3, pp. 685-694, 2025, doi: 10.30998/jrami.v6i03.13983.
- [11] Y. A. Astuti, “Analisis Pengujian Data Algoritma Bubble Sort,” *REMIK: Riset dan E-Jurnal Manajemen Informatika Komputer*, vol. 7, no. 3, pp. 1413-1420, 2023, doi: 10.33395/remik.v7i3.12594.
- [12] A. H. Saptadi and D. W. Sari, “Analisis Algoritma Insertion Sort, Merge Sort dan Implementasinya dalam Bahasa Pemrograman C++,” *Jurnal INFOTEL*, vol. 4, no. 2, pp. 10-17, 2012, doi: 10.20895/infotel.v4i2.103.