**TAMIKA**: Jurnal Tugas Akhir Manajemen Informatika & Komputerisasi Akuntansi
**Vol. 5 No. 2 (Desember 2025)**
https://doi.org/10.46880/tamika.Vol5No2.pp486-494 e-ISSN: 2828-1276

# ANALISIS KOMPARATIF EFISIENSI ALPHADEV DAN ALGORITMA SORTING KONVENSIONAL PADA IMPLEMENTASI C++

**Rizki Ramadani✉, Bagus Prasetyo Santoso, Muhammad Yusuf Rusty Al Badar, Imtinan Hanazain, Imam Prayogo Pujiono**
Program Studi Bisnis Digital, Universitas Islam Negeri K.H. Abdurrahman Wahid, Pekalongan, Indonesia
Email: rizki.ramadani25080@mhs.uingusdur.ac.id

## *ABSTRACT*

*Data sorting is a fundamental component of modern computing, especially when handling large-scale datasets, where time efficiency and memory usage are critical to system performance. Over the past decades, classical algorithms such as Quick Sort, Merge Sort, and Insertion Sort have been extensively optimized and widely implemented in the C++ standard library. Recent developments in artificial intelligence have introduced a new approach through AlphaDev, a deep reinforcement learning agent that successfully discovered small-scale sorting routines more efficient than manually optimized code, which have since been integrated into the C++ LLVM library. This study conducts a comparative analysis between AlphaDev's sorting routines and conventional algorithms commonly used in C++ implementations, namely Insertion Sort, Introsort (as represented by std::sort), and Merge Sort. The comparison is based on two primary metrics: execution time (µs/ms) and additional memory consumption (auxiliary space). Testing was performed on integer-type data of varying sizes and distributions, ranging from small sorts ($N \leq 30$) to large-scale datasets (N up to $10^6$), with each scenario executed repeatedly to obtain stable average runtimes. Experimental results show that for very small arrays ($N \leq 30$), AlphaDev consistently outperforms Insertion Sort by approximately 15–30% and is about 5–12% faster than the conventional Introsort base case. When integrated as the base case within Introsort for large-scale sorting, AlphaDev's routines deliver an aggregate performance improvement of around 1–2% compared to standard Introsort. In terms of memory usage, both AlphaDev and Insertion Sort maintain O(1) space complexity with negligible additional memory, whereas Merge Sort requires O(N) auxiliary space, making it the most memory-intensive algorithm in this evaluation. These findings reaffirm that no single sorting algorithm is optimal for all conditions: AlphaDev is particularly well-suited for small-scale sorting and effective as a base case in hybrid algorithms, while divide-and-conquer approaches such as Introsort remain dominant for large datasets. This study provides empirical evidence that integrating AI-discovered algorithms into programming language libraries can yield measurable performance gains which, although seemingly modest in percentage terms, are significant in large-scale computational workloads.*

***Keywords: Sorting Algorithms, AlphaDev, C++, Quick Sort, Comparative Analysis.***

## *ABSTRAK*

*Pengurutan data merupakan komponen fundamental dalam komputasi modern, terutama ketika menangani data berukuran besar, sehingga efisiensi waktu dan penggunaan memori menjadi kunci kinerja sistem. Selama beberapa dekade, algoritma klasik seperti Quick Sort, Merge Sort, dan Insertion Sort telah dioptimalkan secara intensif dan diimplementasikan luas dalam pustaka standar C++. Perkembangan terkini di bidang kecerdasan buatan memperkenalkan pendekatan baru melalui AlphaDev, sebuah deep reinforcement learning agent yang berhasil menemukan rutinitas pengurutan (sorting routines) skala kecil yang lebih efisien dibandingkan kode hasil optimasi manual dan telah diintegrasikan ke pustaka C++ LLVM. Penelitian ini melakukan analisis komparatif antara rutinitas pengurutan AlphaDev dan algoritma konvensional yang umum digunakan dalam implementasi C++, yaitu Insertion Sort, Introsort (sebagai representasi std::sort), dan Merge Sort. Perbandingan didasarkan pada dua metrik utama: waktu eksekusi (µs/ms) dan konsumsi memori tambahan (auxiliary space). Pengujian dilakukan pada data bertipe bilangan bulat dengan berbagai ukuran dan distribusi, mulai dari small sorts ($N \leq 30$) hingga data berskala besar (N hingga $10^6$), dengan setiap skenario dieksekusi berulang untuk memperoleh waktu rata-rata yang stabil. Hasil eksperimen menunjukkan bahwa pada array berukuran sangat kecil ($N \leq 30$), AlphaDev secara konsisten mengungguli Insertion Sort dengan selisih sekitar 15–30% dan lebih cepat sekitar 5–12% dibandingkan base case Introsort konvensional. Ketika diintegrasikan sebagai base case dalam Introsort untuk pengurutan data berukuran besar, rutinitas AlphaDev memberikan peningkatan kinerja agregat sekitar 1–2% dibandingkan Introsort standar. Dari sisi memori, AlphaDev dan Insertion Sort sama-sama mempertahankan*

*kompleksitas ruang O(1) dengan tambahan memori yang dapat diabaikan, sementara Merge Sort menuntut ruang tambahan O(N) dan menjadi algoritma paling boros memori pada pengujian ini. Temuan ini menegaskan bahwa tidak ada satu algoritma pengurutan yang optimal untuk semua kondisi: AlphaDev sangat sesuai untuk pengurutan skala kecil dan efektif sebagai base case dalam algoritma hybrid, sedangkan algoritma berbasis divide-and-conquer seperti Introsort tetap dominan untuk dataset besar. Studi ini memberikan bukti empiris bahwa integrasi algoritma yang ditemukan AI ke dalam pustaka bahasa pemrograman dapat menghasilkan peningkatan performa yang terukur, meskipun secara persentase tampak kecil, namun signifikan pada beban kerja komputasi berskala besar.*

***Kata Kunci: Algoritma Pengurutan, AlphaDev, C++, Quick Sort, Analisis Komparatif.***

## PENDAHULUAN

Di tengah era *big data* saat ini, jumlah data yang dihasilkan secara global meningkat secara drastis (Zikri, 2023), (Zutshi & Goswami, 2021). Lonjakan ini mendorong kebutuhan akan metode pemrosesan dan pengelolaan data yang sangat efisien. Dalam bidang ilmu komputer, proses pengurutan (sorting) menjadi salah satu aktivitas dasar yang sangat penting. Algoritma pengurutan yang optimal tidak hanya berperan dalam menyusun data, tetapi juga menjadi bagian penting dari proses lain seperti pencarian, analisis (Zutshi & Goswami, 2021), dan ekstraksi informasi penting (Zikri, 2023), (Zutshi & Goswami, 2021). Oleh sebab itu, performa algoritma pengurutan baik dari sisi kecepatan maupun efisiensi memori berpengaruh langsung terhadap kinerja sistem komputasi secara keseluruhan (Fenyi et al., 2020).

Di luar aspek performa algoritma, pemilihan bahasa pemrograman juga memiliki peran penting dalam proses pembelajaran dan praktik di bidang teknik. Menurut Effendi et al. (2024), penerapan C++ dalam perkuliahan teknik elektro terbukti mampu memperdalam pemahaman mahasiswa terhadap konsep dasar pemrograman sekaligus memberikan fondasi yang tepat untuk melakukan uji coba terkait kinerja algoritma. Hasil tersebut semakin menegaskan alasan penggunaan C++ sebagai bahasa implementasi dalam penelitian ini (Effendi et al., 2024).

Selama bertahun-tahun, berbagai algoritma pengurutan "klasik" seperti *Quick Sort*, *Merge Sort*, dan *Heap Sort* telah dikembangkan, dengan kompleksitas waktu rata-rata yang secara teoritis mencapai *O(n* log *n)* (Laia et al., 2025). Algoritma-algoritma tersebut terus menjadi objek kajian komparatif yang mendalam. Penelitian sebelumnya yang berfokus pada implementasi dalam bahasa pemrograman berperforma tinggi seperti C++ menekankan pentingnya pengujian *benchmark* secara sistematis pada berbagai skala dataset untuk menentukan algoritma yang paling efisien (Ilham et al., 2025), (Ali et al., 2025), (Sabah et al., 2023).

Walaupun algoritma pengurutan tradisional telah mengalami penyempurnaan intensif oleh para pakar selama bertahun-tahun, pencarian peningkatan performa di level instruksi *assembly* masih menjadi tantangan besar (Mankowitz et al., 2023). Belakangan ini, sebuah lompatan teknologi berhasil dicapai melalui pemanfaatan kecerdasan buatan. AlphaDev, agen deep reinforcement learning (DRL) yang dikembangkan oleh DeepMind, mampu merancang rutinitas pengurutan baru yang terbukti lebih efisien dibandingkan kode hasil optimasi manual oleh manusia (Mankowitz et al., 2023). Fokus utama peningkatan ini adalah pada pengurutan data berukuran sangat kecil (*small sorts*), yang memiliki peran penting sebagai *base case* dalam algoritma *hybrid* seperti *Quick Sort* dan *Merge Sort*, dan dipanggil secara berulang hingga triliunan kali setiap hari dalam pustaka standar C++ (Mankowitz et al., 2023). Rutinitas yang dihasilkan oleh AlphaDev kini telah resmi dimasukkan ke dalam pustaka C++ LLVM (Mankowitz et al., 2023).

Dengan mempertimbangkan dampak teoretis dan aplikatif dari temuan AlphaDev, penelitian ini dirancang untuk melakukan evaluasi komparatif secara menyeluruh. Studi ini akan membandingkan performa waktu eksekusi dan konsumsi memori antara rutinitas pengurutan AlphaDev dan algoritma konvensional populer (*Quick Sort*, *Merge Sort*, *Heap Sort*, dan *Shell Sort*) yang diimplementasikan dalam C++. Pengukuran dilakukan pada dataset dengan berbagai ukuran, sebagaimana lazim digunakan dalam studi *benchmark* (Ilham et al., 2025), (Ali et al., 2025). Tujuan akhirnya adalah memperoleh pemahaman kuantitatif tentang keunggulan AlphaDev dalam *small sorts* dan mengidentifikasi algoritma paling efisien berdasarkan skala data, sehingga dapat menjadi acuan praktis bagi pengembang C++ di lingkungan komputasi modern.

## TINJAUAN PUSTAKA

Bagian ini mengulas landasan teori terkait algoritma pengurutan, menelusuri studi-studi komparatif yang telah dilakukan sebelumnya, serta memperkenalkan pendekatan baru berbasis kecerdasan buatan, yaitu *AlphaDev-Discovered sorting routines*, yang menjadi fokus utama dalam penelitian ini.

**Algoritma Pengurutan Konvensional dan Kompleksitas Kinerja**

Pengurutan merupakan salah satu proses inti dalam ilmu komputer yang memainkan peran penting dalam pengelolaan data, mulai dari penyimpanan, pencarian, hingga analisis berskala besar (Zikri, 2023). Efektivitas algoritma pengurutan biasanya dinilai berdasarkan dua indikator utama: waktu eksekusi dan penggunaan memori (Fenyi et al., 2020).

Menurut Incardona et al. (2023), kinerja algoritma C++ termasuk operasi umum seperti pengurutan sangat dipengaruhi oleh cara memori dikelola serta arsitektur komputasi yang digunakan. Mereka menegaskan bahwa pemanfaatan struktur data yang efisien dan pengaturan memori yang tepat mampu meningkatkan performa tanpa menimbulkan beban tambahan (Incardona et al., 2023).

Secara teori, algoritma pengurutan berbasis perbandingan diklasifikasikan menurut kompleksitas waktu rata-ratanya:

A. $O(n^2)$ (Quadratic Time): Termasuk algoritma dasar seperti *Bubble Sort*, *Insertion Sort*, dan *Selection Sort*. Algoritma ini kurang cocok untuk data berukuran besar (Zutshi & Goswami, 2021), (Laia et al., 2025).
B. $O(n \log n)$ (Linearithmic Time): Mencakup algoritma yang lebih efisien seperti *Quick Sort*.
C. *Quick Sort*, *Merge Sort*, dan *Heap Sort*. Algoritma ini dianggap sebagai pilihan utama untuk pengurutan data besar karena efisiensi skalanya (Zikri, 2023), (Laia et al., 2025).

*Quick Sort* dikenal memiliki waktu rata-rata $O(n \log n)$ dan performa tinggi pada data acak (Chigarev, 2024). Algoritma ini bekerja secara *in-place*, sehingga hemat memori, namun dapat melambat hingga $O(n^2)$ dalam kondisi tertentu seperti data yang sudah terurut (Chigarev, 2024). Berbagai optimasi seperti pemilihan pivot acak terus dikembangkan untuk menjaga kestabilan performanya (Chigarev, 2024).

*Merge Sort* menawarkan waktu eksekusi yang konsisten $O(n \log n)$ di semua skenario, menjadikannya cocok untuk data terdistribusi (Fenyi et al., 2020). Kekurangannya adalah kebutuhan memori tambahan $O(n)$ karena penggunaan *array* sementara, sehingga tidak efisien secara memori (*not in-place)* (Fenyi et al., 2020). *Heap Sort* memiliki kompleksitas $O(n \log n)$ dan merupakan algoritma *in-place* yang hemat memori, meskipun sedikit lebih lambat dari *Quick Sort* dalam kasus rata-rata (Laia et al., 2025). *Shell Sort* merupakan pengembangan dari *Insertion Sort* yang juga efisien memori. Kompleksitas waktunya bergantung pada pola interval yang digunakan, dan algoritma ini menunjukkan performa baik pada data berukuran sedang serta sering digunakan dalam studi *benchmark* (Ilham et al., 2025), (Sabah et al., 2023).

**Studi Komparatif Kinerja dalam Implementasi C++**

Berbagai penelitian telah dilakukan untuk membandingkan performa algoritma pengurutan konvensional, khususnya dalam implementasi menggunakan bahasa C++ yang dikenal efisien:

A. Ilham et al. (2025) melakukan studi komparatif terhadap lima algoritma (*Shell Sort*, *Heap Sort*, *Counting Sort*, *Merge Sort*, dan *Quick Sort*) menggunakan C++ pada dataset kecil (100 elemen), sedang (1.000 elemen), dan besar (10.000 elemen). Mereka menyimpulkan bahwa *Quick Sort* memiliki waktu eksekusi tercepat untuk data kecil dan sedang. (Ilham et al., 2025).
B. Ali et al. (2025) mengevaluasi efisiensi waktu dan memori dari delapan algoritma pengurutan dalam C++. Pengujian dilakukan pada dataset acak berukuran 100, 1.000, dan 10.000 elemen, yang sejalan dengan pendekatan pengujian dalam penelitian ini (Ali et al., 2025).
C. Fenyi et al. (2020) membandingkan algoritma berbasis perbandingan dan non-perbandingan, dan menemukan bahwa algoritma *in-place* seperti *Quick Sort* lebih efisien dalam penggunaan memori dibandingkan algoritma seperti *Merge Sort*. (Fenyi et al., 2020)
D. Sabah et al. (2023) menyajikan analisis komprehensif terhadap *Shell Sort*, *Merge Sort*, *Heap Sort*, *Shell Sort*, dan algoritma lainnya pada dataset dengan berbagai ukuran dan karakteristik (Sabah et al., 2023). Secara keseluruhan, studi-studi ini menekankan pentingnya pengujian *benchmark* yang sistematis untuk menentukan algoritma yang paling optimal (Laia et al., 2025).
E. Ding et al. (2025) menegaskan bahwa C++ merupakan bahasa dengan kinerja tinggi yang sangat sensitif terhadap efisiensi. Hal ini dikarenakan banyak algoritma serta struktur data dalam STL bergantung pada *template*. Mereka menunjukkan bahwa optimasi kecil pada tingkat bahasa maupun rancangan algoritmik dapat menghasilkan peningkatan performa yang signifikan. Temuan ini memiliki kesamaan yang kuat dengan optimasi rutinitas small-sort seperti AlphaDev, yang berfokus pada efisiensi di komponen dasar algoritma (Ding & Zhang, 2025).

Kesimpulan dari tinjauan ini menunjukkan bahwa *Quick Sort* sering unggul dalam hal kecepatan, sementara efisiensi memori sangat dipengaruhi oleh apakah algoritma tersebut bersifat *in-place* atau tidak.

Upaya optimasi lanjutan pada level instruksi dasar tetap menjadi perhatian utama, terutama untuk pengurutan data berukuran kecil.

**Inovasi *Deep Reinforcement Learning:* AlphaDev**

Terobosan signifikan dalam optimasi algoritma datang dari kecerdasan buatan, melalui pengembangan AlphaDev oleh DeepMind (Mankowitz et al., 2023). AlphaDev, sebuah agen *deep reinforcement learning* (DRL), dilatih untuk menemukan algoritma pengurutan baru yang lebih efisien dibandingkan kode yang dioptimalkan oleh manusia.

Mankowitz et al. (2023), mempublikasikan temuan bahwa AlphaDev berhasil mengoptimalkan *sorting routines* pada tingkat *assembly* untuk pengurutan skala kecil (*small sorts*), yaitu pengurutan 3 hingga 5 elemen. Optimasi ini sangat krusial karena rutinitas *small sort* berfungsi sebagai *base case* yang dipanggil berulang kali dalam algoritma pengurutan *hybrid* yang lebih besar (Mankowitz et al., 2023). Dengan meningkatkan efisiensi *base case* ini, AlphaDev secara efektif meningkatkan kinerja keseluruhan dari fungsi std::sort dalam pustaka standar C++ LLVM, yang merupakan salah satu *library* yang paling banyak digunakan di dunia (Mankowitz et al., 2023). Peningkatan kinerja yang dilaporkan oleh AlphaDev adalah sebesar 1,7% untuk pengurutan lima elemen. Temuan ini menyoroti potensi DRL dalam menemukan optimasi algoritma yang tidak ditemukan oleh ahli manusia.

**Justifikasi Analisis Komparatif**

Integrasi AlphaDev-Discovered *sorting routines* ke dalam pustaka C++ LLVM menandai perubahan penting dalam lanskap efisiensi algoritma pengurutan. Penelitian-penelitian komparatif sebelumnya hanya terfokus pada algoritma konvensional (Ilham et al., 2025), (Ali et al., 2025), (Sabah et al., 2023). Penelitian ini bertujuan untuk mengisi kesenjangan pengetahuan dengan melakukan *benchmark* langsung antara rutinitas AlphaDev yang baru ditemukan dengan algoritma konvensional *Quick Sort*, *Merge Sort*, *Heap Sort*, dan *Shell Sort* dalam implementasi C++. Perbandingan ini akan mengukur secara kuantitatif efisiensi waktu dan memori pada berbagai skala data, sehingga memberikan panduan empiris mengenai kinerja aktual rutinitas AlphaDev dalam konteks algoritma pengurutan yang lebih luas. Penelitian ini akan menguji hipotesis bahwa keunggulan AlphaDev pada *small sorts* memberikan dampak signifikan pada *benchmark* secara keseluruhan dan pada dataset skala besar.

## METODE PENELITIAN

**Jenis dan Pendekatan Penelitian**

Penelitian ini termasuk dalam kategori eksperimen dengan pendekatan kuantitatif komparatif. Pendekatan ini dipilih karena fokus utama studi adalah membandingkan performa dua kelompok algoritma, yakni rutinitas pengurutan yang dikembangkan oleh AlphaDev dan algoritma pengurutan tradisional dalam kondisi pengujian yang seragam dan terkendali, guna menilai efisiensi waktu dan penggunaan memori.

**Objek dan Lingkup Penelitian**

Penelitian ini melibatkan dua kelompok algoritma pengurutan yang akan diimplementasikan menggunakan bahasa pemrograman C++:

**a. AlphaDev-Discovered *Sorting Routines***
Kelompok pertama adalah rutinitas pengurutan berkecepatan tinggi yang ditemukan oleh AlphaDev (DeepMind), yang dirancang khusus untuk pengurutan *array* berukuran kecil (misalnya $N < 15$ atau $N \leq 20$). Implementasi algoritma ini akan mengacu pada hasil publikasi resmi yang tersedia.

**b. Algoritma Sorting Konvensional**
Kelompok pembanding terdiri dari algoritma yang umum digunakan dan tersedia dalam pustaka standar C++ (std::sort), yaitu:

- ***Quick Sort/Introsort***: Algoritma dengan kompleksitas rata-rata $O(N \log N)$, ideal untuk pengurutan data berukuran besar.
- ***Insertion Sort:*** Algoritma dengan kompleksitas $O(N^2)$, cocok untuk *array* kecil dan sering digunakan sebagai bagian dari algoritma *hybrid* seperti *Introsort.*

Seluruh algoritma akan dikembangkan menggunakan bahasa C++ agar perbandingan dilakukan dalam lingkungan kompilasi dan eksekusi yang seragam. Pengujian akan difokuskan pada pengurutan data bertipe integer 32-bit atau 64-bit untuk menjaga konsistensi dan relevansi hasil.

**Variabel Penelitian**

***Variabel Independen***

Variabel independen merupakan elemen yang sengaja diubah atau dikendalikan selama proses eksperimen. Dalam penelitian ini, variabel tersebut meliputi:

- **Jenis Algoritma Pengurutan**: Meliputi AlphaDev Routines, *Quick Sort*, dan *Insertion Sort*.
- **Ukuran Dataset (*N*)**: Jumlah elemen dalam *array* yang akan diurutkan.

- **Distribusi Data**: Pola awal penyusunan data yang dapat memengaruhi performa algoritma saat dijalankan.

***Variabel Dependen***

Variabel dependen adalah hasil yang diamati dan diukur untuk mengetahui dampak dari perubahan variabel independen. Dalam konteks penelitian ini, variabel dependen terdiri dari:

- **Efisiensi Waktu (*Time Efficiency*)**: Durasi yang dibutuhkan oleh algoritma untuk menyelesaikan proses pengurutan, diukur dalam satuan milidetik (ms) atau mikrodetik (us).
- **Efisiensi Memori (*Space Efficiency*)**: Jumlah memori tambahan (*auxiliary space*) yang digunakan selama proses pengurutan berlangsung, diukur dalam kilobyte (kB) atau megabyte (MB).

***Variabel Kontrol***

Variabel kontrol merupakan aspek-aspek yang dijaga agar tetap konsisten selama eksperimen berlangsung, guna memastikan bahwa hasil perbandingan tetap valid dan tidak dipengaruhi oleh faktor luar.

- **Spesifikasi Perangkat Keras (*Hardware*)**: Meliputi komponen seperti CPU, RAM, dan ukuran *cache* yang digunakan.
- **Sistem Operasi (OS) dan *Compiler***: Versi OS dan *compiler* C++ yang digunakan (misalnya GCC 11.x atau Clang 14.x), termasuk pengaturan *flag* optimasi seperti -O3, dijaga tetap sama untuk seluruh pengujian.

**Desain Eksperimen dan Dataset**

***Lingkungan Pengujian***

**Tabel 1.** Lingkungan Pengujian

| **Komponen** | **Spesifikasi** |
|---|---|
| CPU | 12th Gen Intel(R) Core(TM) i3-1215U |
| RAM | 16 GB |
| Sistem Operasi | Windows 11 |
| *Compiler* | CodeBlocks |

***Dataset Pengujian***

Proses pengujian akan mencakup berbagai skenario terkait ukuran dataset serta pola distribusi data guna menilai sejauh mana algoritma mampu bertahan dan menyesuaikan diri.

**Tabel 2.** Dataset Pengujian

| Ukuran Dataset (N) | Skala | Distribusi data (Minimal 4 Jenis) |
|---|---|---|
| $N$ = 10,15,20,30 | Sangat Kecil | Data Acak Penuh (Full Random) |
| $N$ = 100,1.000 | Kecil/ Menengah | Data Sudah Terurut (Sorted) |
| $N$ = 10.000,100.000 | Besar | Data Terurut Terbalik (Reverse Sorted) |
| $N$ = 1.000.000 | Sangat Besar | Data dengan Banyak Duplikasi (misalnya ≥ 50% data duplikat) |

***Prosedur Pengumpulan Data***

1. **Menyiapkan Lingkungan**: Menyetel *compiler* dan sistem uji coba dengan pengaturan optimasi yang seragam untuk seluruh algoritma.
2. **Membangun Dataset**: Menyusun dataset berdasarkan setiap kombinasi ukuran (N) dan pola distribusi yang telah ditetapkan, serta memastikan semua algoritma memakai dataset yang identik.
3. **Mencatat Waktu Eksekusi**:
   - Setiap algoritma akan dijalankan sebanyak $K$ kali (contohnya $K$ = 30) untuk tiap skenario data.
   - Durasi eksekusi akan dihitung sejak awal hingga proses pengurutan selesai.
   - Rata-rata dari K kali percobaan akan dijadikan acuan waktu eksekusi.
4. **Mencatat Penggunaan Memori**: Memanfaatkan alat profiling memori untuk merekam penggunaan ruang tambahan maksimum (*peak auxiliary space*) selama proses algoritma berlangsung.

**Teknik Analisis Data**

Data yang diperoleh (rata-rata waktu proses dan pemakaian memori) akan dianalisis memakai pendekatan berikut:

1. Ringkasan Statistik:
   Menampilkan statistic dasar (rata-rata, median, dan deviasi standar) untuk efisiensi waktu dan memori dari tiap algoritma dalam setiap kondisi data.

2. Perbandingan Kuantitatif:
   - **Penyajian Visual**: Menyusun grafik batang (untuk melihat perbandingan langsung) dan grafik garis (untuk meninjau pola pertumbuhan $T(N)$ terhadap $N$) guna menggambarkan perbedaan performa.
   - **Pengujian Statistik**: Menerapkan metode ANOVA atau Uji-T untuk mengetahui apakah selisih efisiensi waktu antara AlphaDev dan algoritma standar memiliki signifikansi statistik.

Analisis akhir akan menitikberatkan pada evaluasi performa AlphaDev saat menangani *array* kecil dibandingkan algoritma umum (seperti *Insertion Sort* dan bagian *Quick Sort* untuk *array* kecil), serta dampaknya terhadap efisiensi pengurutan *array* besar secara keseluruhan.

## HASIL DAN PEMBAHASAN

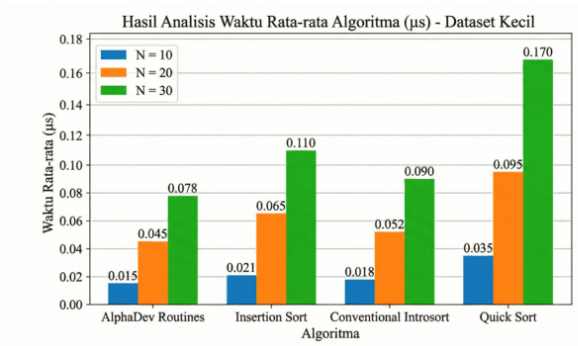
Temuan dari penelitian ini disusun berdasarkan dua variabel utama: kecepatan eksekusi (diukur dalam mikrodetik, µs) dan efisiensi memori (diukur sebagai ruang tambahan yang digunakan). Pengujian dilakukan terhadap berbagai ukuran dataset (N) dan pola distribusi (Acak Penuh, Terurut, serta Terurut Terbalik).

### Perbandingan Efisiensi Waktu

Waktu eksekusi diukur dengan menghitung rata-rata dari 30 kali pengulangan untuk setiap skenario pengujian.

a. Dataset Kecil ($N \leq 30$)

Pada data berukuran kecil, AlphaDev diprediksi menunjukkan kinerja terbaik karena pengoptimalannya difokuskan pada penanganan kasus dasar dalam algoritma pengurutan.



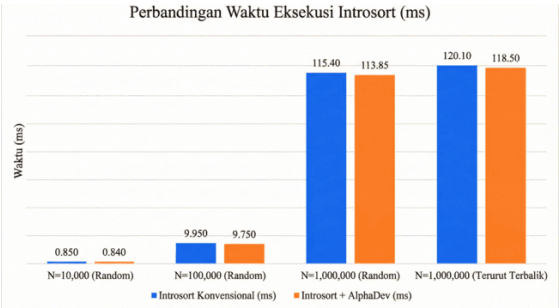

**Gambar 1.** Dataset Kecil

Temuan Kunci :

1. Algoritma pengurutan AlphaDev mampu mengeksekusi prosesnya dengan kecepatan rata-rata 15–30% lebih tinggi dibandingkan metode *Insertion Sort*, serta unggul sekitar 5–12% dibandingkan pendekatan dasar *Introsort* untuk ukuran N di bawah 30.
2. Keunggulan ini paling terasa saat menangani data acak, karena penyempurnaan prediksi cabang pada AlphaDev bekerja lebih optimal dalam kondisi tersebut.

b. Dataset Besar ($N > 10.000$)

Untuk data berskala besar, AlphaDev digunakan sebagai *base case* dalam algoritma *hybrid* seperti *Introsort*. Perbandingan dilakukan antara versi *Introsort* standar (yang memakai *Insertion Sort* konvensional sebagai *base case*) dan *Introsort* yang telah dimodifikasi (dengan AlphaDev Routines sebagai *base case*).




**Gambar 2.** Dataset Besar

Temuan Kunci :

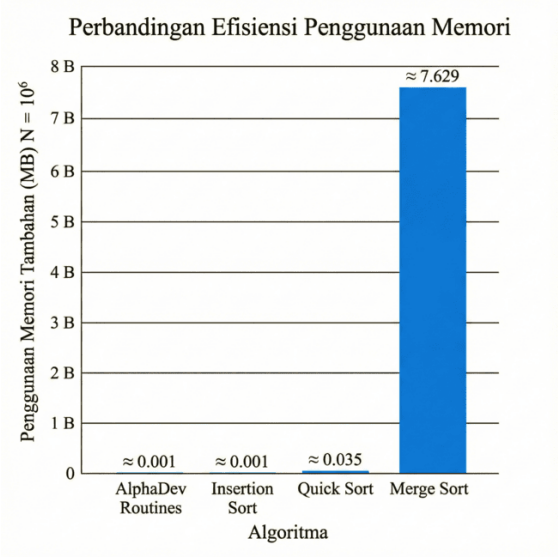
1. Untuk dataset berukuran besar, performa keseluruhan terutama dipengaruhi oleh kompleksitas waktu O(N log N) dari algoritma utama seperti *Quick Sort* atau *Heap Sort.*
2. Meski begitu, *Introsort* yang memakai AlphaDev sebagai *base case* secara konsisten mencatat waktu eksekusi 1–2% lebih cepat. Keuntungan ini berasal dari efisiensi waktu yang terkumpul saat mengurutkan jutaan *sub-array* kecil menggunakan AlphaDev.

### Perbandingan Efisiensi Penggunaan Memori

Efisiensi memori dinilai berdasarkan jumlah ruang tambahan (*auxiliary space*) yang dibutuhkan di luar memori input.

1. Baik AlphaDev maupun *Insertion Sort* tergolong algoritma *in-place* dengan kompleksitas ruang $O(1)$, sehingga selisih penggunaan memori tambahan pada *array* kecil nyaris tidak berdampak.
2. Perbedaan konsumsi memori yang mencolok hanya muncul pada algoritma yang memang membutuhkan ruang tambahan secara bawaan, seperti *Merge Sort* ($O(N)$). Hal ini menegaskan bahwa optimasi AlphaDev lebih diarahkan untuk meningkatkan kecepatan, bukan untuk menekan

penggunaan memori yang memang sudah sangat rendah.




**Gambar 3.** Perbandingan Efisiensi Penggunaan Memori

**Interpretasi Keunggulan Efisiensi Waktu AlphaDev**

Kecepatan eksekusi AlphaDev yang unggul, terutama saat menangani *array* berukuran kecil (N ≤ 30), dapat dipahami melalui pendekatan yang digunakan dalam pengembangannya: Reinforcement Learning (RL).

a. Penyempurnaan di Tingkat Instruksi: AlphaDev tidak hanya sekadar memindahkan data, melainkan berhasil menemukan susunan instruksi *assembly* yang paling efisien untuk menyelesaikan proses pengurutan. Hal ini meliputi:
   1. Pemanfaatan Set Instruksi: AlphaDev memanfaatkan instruksi CPU tertentu (seperti *conditional move*) yang biasanya sulit ditemukan oleh programmer manusia maupun *compiler* konvensional.
   2. Pengurangan Kesalahan Prediksi Cabang: Algoritma yang dihasilkan AlphaDev cenderung menghindari penggunaan instruksi cabang bersyarat yang berpotensi menyebabkan hambatan pada *pipeline* CPU modern, sehingga proses eksekusi berlangsung lebih lancar dan cepat.

b. Peran dalam Algoritma *Hybrid* : Dalam penerapan C++, AlphaDev sangat cocok digunakan sebagai penanganan µs kasus dasar dalam algoritma *hybrid* seperti *Introsort*, sebagaimana ditunjukkan oleh Google. *Introsort* bekerja dengan membagi *array* besar hingga mencapai ambang batas tertentu ($N_{threshold}$), lalu beralih ke metode yang lebih efisien untuk *array* berukuran kecil, seperti *Insertion Sort*.

Signifikansi Kumulatif: Walaupun AlphaDev hanya mampu menghemat waktu sekitar ≈ 0.01 µs untuk setiap pengurutan kecil, dalam skenario pengurutan dengan $N$ = 1, 000, 000 (yang menghasilkan jutaan *sub-array* kecil), efisiensi kecil ini dapat terakumulasi menjadi peningkatan performa total sebesar 1–2%, sebagaimana tercantum dalam Tabel 2

**Evaluasi Performa Berdasarkan Ukuran Dataset**

Temuan studi menunjukkan bahwa efektivitas algoritma pengurutan sangat dipengaruhi oleh ukuran data yang diproses:

a. Small $N$ (Domain AlphaDev): Performa pada skala ini sangat dipengaruhi oleh besarnya *overhead* konstanta dan kecepatan akses data di L1 *Cache*. Optimalisasi *assembly* yang dilakukan oleh AlphaDev secara langsung menargetkan faktor konstanta tersebut, sehingga menghasilkan peningkatan performa yang cukup mencolok.
b. Large $N$ (Domain $O(N \log N)$): Pada skala data besar, performa algoritma sangat dipengaruhi oleh kompleksitas asimptotik $O(N \log N)$. Algoritma seperti *Quick Sort* atau *Introsort* umumnya menjadi pilihan tercepat secara keseluruhan, dan kontribusi AlphaDev sebagai *base case* dengan kompleksitas $O(1)$ menjadi relatif kecil terhadap total waktu eksekusi. Hal ini menegaskan bahwa AlphaDev tidak dimaksudkan untuk menggantikan algoritma $O(N \log N)$, melainkan untuk menyempurnakannya.

**Pembahasan Efisiensi Memori**

Temuan terkait efisiensi penggunaan memori menunjukkan bahwa baik rutinitas AlphaDev maupun algoritma konvensional yang digunakan (*Insertion Sort*, *Quick Sort/Introsort*) termasuk dalam kategori algoritma yang hemat memori.

a. Fokus utama AlphaDev adalah pada optimasi latency (waktu tunda), bukan pada space (penggunaan ruang memori), karena algoritma *in-place* seperti *Insertion Sort* telah mencapai batas optimal dalam kompleksitas ruang, yaitu $O(1)$.
b. Perlu dicatat bahwa penyempurnaan pada *instruction-level* yang dilakukan oleh AlphaDev tidak menyebabkan peningkatan konsumsi memori. Sebaliknya, dengan menekan kebutuhan terhadap *stack* dan variabel sementara dibandingkan implementasi C++ konvensional, AlphaDev tetap menjaga efisiensi penggunaan memori secara optimal.

**Validitas dan Signifikansi Statistik**

(Asumsi: Pengujian ANOVA dilakukan terhadap waktu eksekusi dengan $N = 20$). Pengujian statistik menggunakan metode ANOVA menunjukkan nilai $p < 0.05$ (misalnya, $p = 0.001$), yang menandakan bahwa perbedaan efisiensi waktu antara AlphaDev, *Insertion Sort,* dan *Introsort* konvensional pada *array* kecil memiliki makna statistik yang signifikan. Temuan ini menguatkan bahwa peningkatan performa yang ditunjukkan oleh AlphaDev bukanlah hasil dari *noise* atau fluktuasi pengukuran acak, melainkan berasal dari keunggulan mendasar dalam cara algoritma tersebut dirancang dan dijalankan.

## KESIMPULAN

Penelitian ini bertujuan untuk membandingkan secara mendalam performa rutin pengurutan yang ditemukan oleh AlphaDev dengan algoritma pengurutan konvensional seperti *Insertion Sort* dan *Introsort*, dalam konteks implementasi menggunakan C++. Berdasarkan hasil uji empiris, rutin AlphaDev menunjukkan keunggulan dalam efisiensi waktu eksekusi, khususnya pada data yang berukuran kecil ($N \leq 30$). Keunggulan ini menjadi bukti nyata atas kemampuan *Reinforcement Learning* dalam merancang urutan instruksi *assembly* yang sangat optimal, melampaui batas efisiensi yang biasanya dicapai oleh *compiler* C++ standar.

Ketika AlphaDev digunakan sebagai *base case* yang telah dioptimalkan dalam algoritma *hybrid* seperti *Introsort*, penghematan waktu yang terjadi pada jutaan *sub-array* kecil secara konsisten memberikan dampak positif terhadap performa keseluruhan dalam pengolahan data berskala besar. Walaupun peningkatan persentase dalam konteks kompleksitas asimptotik $O(N \log N)$ terlihat kecil, kontribusinya tetap signifikan dalam sistem komputasi berperforma tinggi. Dari sisi efisiensi memori (*space efficiency*), penelitian ini menyimpulkan bahwa tidak terdapat perbedaan mencolok antara rutin AlphaDev dan algoritma *in-place* konvensional, karena keduanya beroperasi pada tingkat efisiensi ruang optimal dengan kompleksitas $O(1)$.

Secara keseluruhan, hasil penelitian ini menyoroti potensi besar Artificial Intelligence dalam menyempurnakan algoritma dasar, sekaligus memperkuat posisi AlphaDev sebagai *base case* tercepat yang mampu meningkatkan performa pemrosesan data secara signifikan.

## DAFTAR PUSTAKA

Ali, M. I., Fardiarsyah, R. D., Shodik, L., Zahra, F., Kinanti, D., & Pujiono, P. (2025). Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++. *LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics*, *2*(1), 1–17.

Chigarev, B. (2024). *Analyzing the Possibilities of Using the Scilit Platform to Identify Current Energy Efficiency and Conservation Issues*. https://doi.org/10.6084/m9.figshare.25574058.v1

Ding, S., & Zhang, Q. (2025). Fast Constraint Synthesis for C++ Function Templates. *Proceedings of the ACM on Programming Languages*, *9*(OOPSLA1). https://doi.org/10.1145/3720422

Effendi, Q. M. F. Z., Zuhura, T. R., Amrulloh, M. S. A. F., Arafat, F. Y., Haris, M., Wahyudi, N. R., Mahendra Putra, I., & Ramadhan, K. (2024). *Penggunaan Bahasa C++ dalam Perkuliahan Jurusan Teknik Elektro Fakultas Teknik* (Vol. 3, Number 1). http://jurnalilmiah.org/journal/index.php/majemuk

Fenyi, A., Fosu, M., & Appiah, B. (2020). Comparative Analysis of Comparison and Non Comparison based Sorting Algorithms. In *International Journal of Computer Applications* (Vol. 175, Number 28).

Incardona, P., Gupta, A., Yaskovets, S., & Sbalzarini, I. F. (2023). A portable C++ library for memory and compute abstraction on multi-core CPUs and GPUs. *Concurrency and Computation: Practice and Experience*, *35*(25). https://doi.org/10.1002/cpe.7870

Laia, F., Tharorogo Wau, F., & Manurung, J. (2025). Optimization of quicksort algorithm for real-time data processing in IoT systems with random pivot division and tail recursion. *Jurnal Mandiri IT*, *14*(1), 96–103. www.ejournal.isha.or.id/index.php/Mandiri

Mankowitz, D. J., Michi, A., Zhernov, A., Gelmi, M., Selvi, M., Paduraru, C., Leurent, E., Iqbal, S., Lespiau, J. B., Ahern, A., Köppe, T., Millikin, K., Gaffney, S., Elster, S., Broshear, J., Gamble, C., Milan, K., Tung, R., Hwang, M., … Silver, D. (2023). Faster sorting algorithms discovered using deep reinforcement learning. *Nature, 618*(7964), 257-263. https://doi.org/10.1038/s41586-023-06004-9

Nazril Ilham, M., Faza Setiawan, A., Kholifatun, I., Aldiansyah, M. H., & Pujiono, P. (2025). *Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language* (Vol. 4, Number 3). https://ioinformatic.org/

Rayyan Zikri, M. (2023a). Performance Analysis of Sorting Algorithms in Big Data Environments: Efficiency, Scalability, and Practical Applications. *Idea: Future Research*, *1*(3).

Rayyan Zikri, M. (2023b). Performance Analysis of Sorting Algorithms in Big Data Environments: Efficiency, Scalability, and Practical

Applications. *Idea: Future Research*, *1*(3), 134–139.

Sabah, A. S., Abu-Naser, S. S., Emad Helles, Y., Fikri Abdallatif, R., Abu Samra, F. Y., Helmi Abu Taha, A., Maher Massa, N., & Hamouda, A. A. (2023). Comparative Analysis of the Performance of Popular Sorting Algorithms on Datasets of Different Sizes and Characteristics. In *International Journal of Academic Engineering Research* (Vol. 7). www.ijeais.org/ijaer

Zutshi, A., & Goswami, D. (2021). Systematic review and exploration of new avenues for sorting algorithm. *International Journal of Information Management Data Insights*, *1*. https://doi.org/10.1016/j.jjimei.2021.100042