Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

Performance Evaluation of a Hybrid Quick Sort-Insertion Sort Algorithm

Aulya Deswita[1)], Nur Ayda Safitri[2)], Makaila Rif'ah Juliana Azzahra[3)], Fitriani Lestari[4)], Imam Prayogo Pujiono[5)]

1. Informatika, Fakultas Sains Dan Teknologi, UIN K.H Abdurrahman Wahid, Indonesia
2. Informatika, Fakultas Sains Dan Teknologi, UIN K.H Abdurrahman Wahid, Indonesia
3. Informatika, Fakultas Sains Dan Teknologi, UIN K.H Abdurrahman Wahid, Indonesia
4. Informatika, Fakultas Sains Dan Teknologi, UIN K.H Abdurrahman Wahid, Indonesia
5. Informatika, Fakultas Sains Dan Teknologi, UIN K.H Abdurrahman Wahid, Indonesia

Article Info
Kata Kunci : *Hybrid Quick-Insertion sort*; *Insertion sort*; *Quick Sort*; Algoritma Penyortiran

***Keywords*:** *Hybrid Quick-Insertion sort; Insertion sort; Quick Sort; Sorting Algorithms*

Article history:
Received: 11 Juni 2026
Revised: 13 Juni 2026
Accepted: 14 Juni 2026
Available: 1 November 2026

DOI :
10.48144/suryainformatika.v16i2.2470

* Corresponding author.
Makaila Rif'ah Juliana Azzahra
E-mail address:
makaila.rifah.juliana25026@mhs.uin-gusdur.ac.id

ABSTRAK
Perkembangan teknologi informasi yang sangat cepat memerlukan adanya metode pengolahan data yang cepat, efisien, dan tepat, terutama dalam proses pengelompokan data dalam jumlah besar. Penelitian ini bertujuan untuk mempelajari bagaimana algoritma *Hybrid Quick-Insertion Sort,* yang merupakan gabungan dari *Quick Sort* dan *Insertion Sort*, bekerja dalam meningkatkan kecepatan dan efisiensi pengurutan data dibandingkan dengan metode pengurutan yang biasa digunakan. Metode yang digunakan dalam penelitian ini adalah eksperimen komputasional dengan menggunakan bahasa pemrograman C++. Algoritma yang dikembangkan diterapkan pada tiga ukuran data, yaitu 100, 1.000, dan 5.000 elemen, untuk mengukur waktu yang diperlukan dalam setiap kondisi pengujian. Hasil pengujian menunjukkan bahwa algoritma *Hybrid Quick-Insertion Sort* secara terus-menerus memberikan waktu pemrosesan yang lebih singkat dan konsisten dibandingkan metode pengurutan lainnya, di mana keunggulannya semakin jelas terlihat ketika mengolah data dalam jumlah besar. Ini menunjukkan bahwa dengan menggabungkan kedua algoritma tersebut, kelemahan dari masing-masing algoritma bisa diatasi. Misalnya, *Quick Sort* memiliki performa yang menurun saat menangani data berukuran kecil, sementara *Insertion Sort* kurang efektif ketika menangani data berukuran besar. Dengan demikian, bisa disimpulkan bahwa algoritma *Hybrid Quick-Insertion Sort* merupakan cara mengurutkan data yang efektif dan efisien, serta memiliki potensi untuk diterapkan dalam berbagai sistem yang membutuhkan pengolahan data dalam volume besar secara optimal.

ABSTRACT
The rapid development of information technology requires fast, efficient, and precise data processing methods, especially in the process of grouping large amounts of data. This study aims to study how the Hybrid Quick-Insertion Sort algorithm, which is a combination of Quick Sort and Insertion Sort, works to improve the speed and efficiency of data sorting compared to commonly used sorting methods. The method used in this study is a computational experiment using the C++ programming language. The developed algorithm was applied to three data sizes, namely 100, 1,000, and 5,000 elements, to measure the time required under each test condition. The test results show that the Hybrid Quick-Insertion Sort algorithm consistently provides shorter and consistent processing times than other sorting methods, where its advantages are increasingly apparent when processing large amounts of data. This suggests that by combining the two algorithms, the weaknesses of each algorithm can be overcome. For example, Quick Sort has a decreased performance when handling small data sizes, while Insertion Sort is less effective when handling large data sizes. Thus, it can be concluded that the Hybrid Quick-Insertion Sort algorithm is an effective and efficient way to sort data, and has the potential to be applied in various systems that require optimal processing of large volumes of data.

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

1. PENDAHULUAN

Perkembangan teknologi informasi yang semakin pesat menyebabkan volume data yang harus diproses oleh sistem komputer terus meningkat. Salah satu proses fundamental dalam pengolahan data adalah pengurutan data (*sorting*), yaitu proses menyusun data ke dalam urutan tertentu untuk mempermudah pencarian, analisis, dan pengolahan lebih lanjut. Berbagai algoritma pengurutan telah dikembangkan, seperti *Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, dan Quick Sort,* yang memiliki karakteristik serta kompleksitas waktu yang berbeda-beda [1].

Dalam implementasinya, algoritma pengurutan konvensional sering menghadapi kendala ketika digunakan pada dataset berukuran besar. Algoritma dengan kompleksitas waktu $O(n^2)$, seperti *Bubble Sort dan Selection Sort*, cenderung membutuhkan waktu eksekusi yang lebih lama dibandingkan algoritma dengan kompleksitas O(n log n), seperti *Merge Sort dan Quick Sort* [2]. Oleh karena itu, pemilihan algoritma pengurutan yang tepat menjadi faktor penting dalam meningkatkan efisiensi komputasi dan kinerja sistem.

Berbagai penelitian telah dilakukan untuk meningkatkan performa algoritma pengurutan melalui pendekatan hybrid sorting, yaitu teknik yang menggabungkan dua atau lebih algoritma pengurutan guna memanfaatkan keunggulan masing-masing algoritma. Penelitian yang dilakukan oleh Al Rivan menunjukkan bahwa kombinasi *Quick Sort dan Merge Sort* dengan *Insertion Sort* mampu menghasilkan performa yang lebih baik dibandingkan penggunaan algoritma tunggal [3]. Penelitian lain juga menunjukkan bahwa kombinasi *Quick-Insertion Sort dan Merge-Insertion Sort* memberikan peningkatan efisiensi waktu eksekusi pada proses pengurutan data [4].

Selain itu, Atmaja dan Pinaryanto mengembangkan *Selection Sort Hybrid* yang mampu meningkatkan efisiensi proses pengurutan melalui mekanisme pencarian nilai minimum dan maksimum secara bersamaan [5]. Penelitian Aryanto dkk. kemudian mengombinasikan *Selection Sort Hybrid* dengan *Bucket Sort* dan memperoleh peningkatan performa pada proses pengurutan data numerik [6].

Secara teoritis, kombinasi Quick Sort dan Insertion Sort merupakan salah satu pendekatan hybrid yang banyak digunakan karena kedua algoritma memiliki karakteristik yang saling melengkapi. Quick Sort menggunakan pendekatan divide and conquer yang menghasilkan kompleksitas waktu rata-rata O(n log n), sehingga sangat efisien untuk menangani dataset berukuran besar. Namun, ketika ukuran partisi menjadi sangat kecil, biaya pemanggilan fungsi rekursif dan proses partisi dapat menjadi lebih besar dibandingkan pekerjaan pengurutannya sendiri. Sebaliknya, Insertion Sort memiliki kompleksitas waktu $O(n^2)$, tetapi bekerja sangat efisien pada dataset berukuran kecil karena tidak memerlukan rekursi, memiliki overhead yang rendah, serta mampu mengurutkan data yang telah atau hampir terurut dengan cepat. Oleh karena itu, penggunaan Quick Sort untuk partisi besar dan Insertion Sort untuk partisi kecil diharapkan dapat mengurangi overhead komputasi sekaligus mempertahankan efisiensi proses pengurutan secara keseluruhan.

Oleh karena itu, algoritma *Hybrid Quick-Insertion Sort* memanfaatkan *Quick Sort* untuk mempartisi data berukuran besar dan menggunakan *Insertion Sort* untuk mengurutkan sub-array berukuran kecil sehingga diharapkan mampu menghasilkan performa yang lebih optimal.

Meskipun beberapa penelitian telah mengkaji kombinasi *Quick Sort* dan *Insertion Sort* serta menunjukkan peningkatan performa dibandingkan algoritma tunggal, sebagian besar penelitian tersebut berfokus pada pengembangan algoritma *hybrid* dan pembuktian efektivitasnya secara umum. Penelitian terdahulu belum banyak menyajikan evaluasi komparatif yang secara langsung membandingkan kinerja *Hybrid Quick-Insertion Sort* dengan algoritma penyusunnya, yaitu *Quick Sort* dan *Insertion Sort*, menggunakan variasi ukuran dataset dalam lingkungan pengujian yang sama. Akibatnya, belum tersedia informasi yang memadai mengenai besarnya peningkatan performa yang diperoleh dari pendekatan hybrid pada setiap skala data. Oleh karena itu, diperlukan penelitian yang secara sistematis mengevaluasi *efektivitas Hybrid Quick-Insertion Sor*t dibandingkan algoritma penyusunnya pada berbagai ukuran dataset.

Selain itu, informasi mengenai sejauh mana pendekatan *hybrid* memberikan keuntungan performa pada kondisi data yang beragam belum dijelaskan secara komprehensif. Kondisi tersebut menunjukkan adanya kebutuhan untuk melakukan evaluasi yang lebih mendalam terhadap efektivitas algoritma *Hybrid Quick-Insertion Sort* pada berbagai skenario pengujian. Berdasarkan permasalahan tersebut, penelitian ini bertujuan untuk menganalisis dan membandingkan kinerja *Quick Sort, Insertion Sort, dan Hybrid Quick-Insertion Sort* pada berbagai ukuran dataset. Evaluasi dilakukan berdasarkan parameter waktu eksekusi dan performa proses pengurutan untuk mengidentifikasi efektivitas masing-masing algoritma dalam menangani jumlah data yang berbeda.

Kontribusi penelitian ini terletak pada penyajian evaluasi komparatif yang sistematis antara *Quick Sort, Insertion Sort, dan Hybrid Quick-Insertion Sort* berdasarkan parameter waktu eksekusi pada dataset berukuran 100, 1.000, dan 5.000 elemen dalam lingkungan pengujian yang sama. Evaluasi ini dilakukan untuk mengidentifikasi tingkat peningkatan performa yang diperoleh dari pendekatan hybrid

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

dibandingkan algoritma penyusunnya serta memberikan gambaran yang lebih jelas mengenai kondisi di mana algoritma hybrid memberikan keuntungan komputasi yang signifikan. Hasil penelitian diharapkan dapat menjadi referensi dalam pemilihan algoritma pengurutan yang lebih efisien untuk mendukung berbagai aplikasi pengolahan data.

2. METODE PENELITIAN

Penelitian menggunakan pendekatan eksperimental komputasional, Dengan menggunakan Objek penelitian tiga algoritma penyortiran: *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort*, Variabel dalam penelitian ini terdiri dari variabel independen, variabel dependen, dan variabel control, Penelitian ini dilakukan menggunakan bahasa pemrograman C++, Prosedur pengujian dalam studi ini mengikuti aturan pengujian algoritma empiris yang dirumuskan oleh McGeoch.

2.1. Jenis Penelitian

Penelitian ini menggunakan pendekatan eksperimental komputasional, yaitu metode penelitian yang berfokus pada pengujian dan perbandingan kinerja algoritma secara empiris melalui simulasi program. Pendekatan ini dipilih karena memungkinkan pengukuran langsung terhadap variabel waktu eksekusi algoritma dalam kondisi yang terkendali dan dapat direplikasi [7]. Penelitian eksperimental dalam ilmu komputer umumnya digunakan untuk mengevaluasi efisiensi algoritma berdasarkan parameter yang dapat diukur, seperti waktu komputasi dan kompleksitas ruang [8].

2.2. Objek Penelitian

Objek penelitian ini adalah tiga algoritma penyortiran: *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort*. *Quick Sort* adalah algoritma bagi dan taklukkan yang membagi sebuah array berdasarkan nilai pivot, dengan kompleksitas waktu rata-rata O(n log n) [9]. *Insertion sort* adalah algoritma sederhana yang bekerja optimal pada dataset kecil dengan kompleksitas terburuk $O(n^2)$ [10]. *Hybrid Quick-Insertion sort*, di sisi lain, adalah pendekatan kombinatorial yang secara spesifik menggabungkan *Quick Sort* dan *Insertion sort*: *Quick Sort* digunakan untuk membagi data secara rekursif, kemudian beralih ke *Insertion sort* ketika ukuran partisi mencapai nilai ambang batas (*threshold*) tertentu. Pendekatan ini dirancang untuk meningkatkan efisiensi penyortiran secara keseluruhan dengan memanfaatkan keunggulan masing-masing algoritma [11].

2.3. Variabel Penelitian

Variabel dalam penelitian ini terdiri dari variabel independen, variabel dependen, dan variabel kontrol, sebagaimana dijelaskan di bawah ini:

- Variabel Independen: Jenis algoritma penyortiran yang digunakan, yaitu *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort*; serta ukuran data masukan (100, 1.000, dan 5.000 elemen).
- Variabel Tergantung: Waktu eksekusi algoritma (dalam detik) yang diukur selama proses penyortiran.
- Variabel Kontrol: Kondisi perangkat keras (CPU, RAM), bahasa pemrograman yang digunakan (C++), dan nilai ambang batas dalam algoritma *Hybrid Quick-Insertion sort*.

2.4. Alat dan Bahan Penelitian

Penelitian ini dilakukan menggunakan bahasa pemrograman C++, yang dipilih karena kemampuannya mengelola memori secara langsung dan kecepatan eksekusinya yang tinggi, menjadikannya pilihan ideal untuk pengujian algoritma [12]. Pengukuran waktu eksekusi dilakukan menggunakan fungsi bawaan dari pustaka <chrono> dalam C++, yang dapat merekam durasi proses dengan presisi milidetik. Program ini dikompilasi menggunakan kompiler GCC dengan optimisasi standar, dan pengujian dilakukan pada sistem operasi berbasis Windows.

2.5. Desain Algoritma

Desain algoritma *Hybrid Quick-Insertion sort* dalam penelitian ini didasarkan pada konsep yang dikembangkan oleh Sedgewick & Wayne [11], yang menyatakan bahwa menggabungkan *Quick Sort* dan *Insertion sort* dapat menghasilkan kinerja yang lebih baik daripada menggunakan salah satu algoritma tersebut secara terpisah. Prinsip utamanya melibatkan penggunaan *Quick Sort* sebagai mekanisme untuk membagi data awal secara rekursif, kemudian beralih ke *Insertion sort* begitu ukuran partisi mencapai nilai threshold tertentu.

Nilai threshold merupakan parameter kritis yang menentukan titik transisi antara kedua algoritma. Memilih threshold yang terlalu kecil akan mengurangi manfaat *Insertion sort*, sementara nilai yang terlalu besar dapat meningkatkan beban rekursif pada *Quick Sort*. Berdasarkan rekomendasi Sedgewick & Wayne [11] serta studi-studi sebelumnya yang menyarankan rentang 10–20 elemen [13], penelitian ini menetapkan nilai threshold sebesar 10 elemen sebagai parameter tetap yang dikontrol selama seluruh pengujian.

2.6. Prosedur Pengujian

Prosedur pengujian dalam studi ini mengikuti aturan pengujian algoritma empiris yang

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

dirumuskan oleh McGeoch [14], dengan tahapan sebagai berikut:

- Persiapan Data: Data uji, yang terdiri dari array bilangan bulat acak, dihasilkan secara otomatis oleh program menggunakan generator bilangan acak bawaan C++. Data diuji pada tiga ukuran berbeda yaitu 100, 1.000, dan 5.000 elemen untuk mengamati perilaku algoritma pada skala data yang bervariasi.
- Implementasi Program: Ketiga algoritma diimplementasikan dalam satu program C++ terintegrasi. Setiap algoritma dikapsulkan dalam fungsi tersendiri sehingga dapat dipanggil secara independen untuk memastikan objektivitas pengujian.
- Pengukuran Waktu Eksekusi: Waktu eksekusi dicatat menggunakan timer yang dimulai tepat sebelum fungsi penyortiran dipanggil dan dihentikan segera setelah proses selesai. Pengukuran waktu dilakukan untuk setiap skenario pengujian guna memastikan konsistensi data.
- Analisis dan Perbandingan: Hasil waktu eksekusi ketiga algoritma dibandingkan untuk setiap ukuran data. Analisis berfokus pada perbedaan efisiensi antar algoritma dan pola peningkatan waktu eksekusi seiring bertambahnya jumlah data.

2.7. Teknik Analisis Data

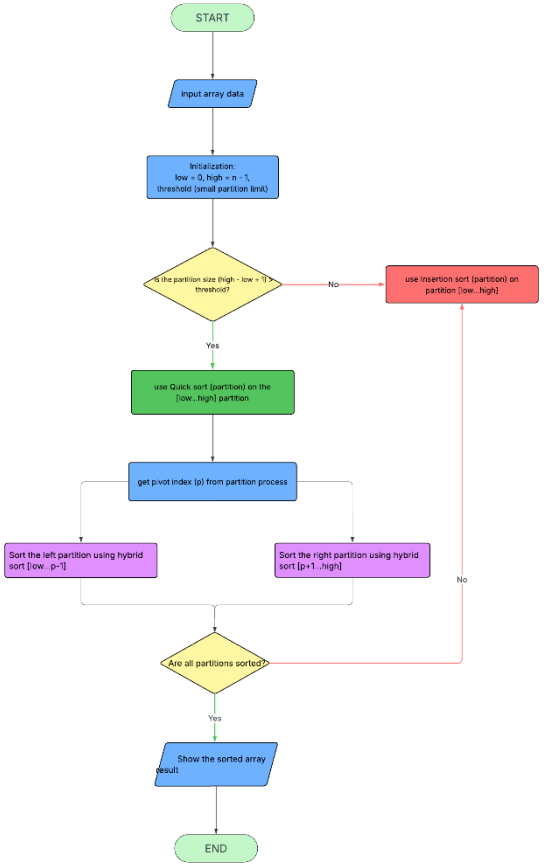
Analisis data dilakukan menggunakan pendekatan deskriptif-komparatif, dengan menyajikan hasil pengujian dalam bentuk tabel dan grafik. Perbandingan kinerja antara *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort* didasarkan pada waktu eksekusi aktual yang dihasilkan oleh program-program tersebut. Selain itu, hasil eksperimen diinterpretasikan dalam kerangka teori kompleksitas algoritma, khususnya notasi Big-O sebagaimana dijelaskan oleh Cormen dkk. [8], untuk memberikan landasan teoretis bagi temuan empiris.

Grafik perbandingan disajikan untuk memvisualisasikan tren waktu eksekusi seiring bertambahnya jumlah data. Visualisasi ini berguna untuk mengidentifikasi algoritma mana yang paling efisien pada setiap skala data dan memberikan gambaran yang lebih intuitif mengenai kompleksitas praktis masing-masing algoritma [15].

2.8. Perancangan dan implementasi sistem

- Flowchart Algoritma

Flowchart pada sistem ini menggambarkan proses kerja



Gambar 1. Flowchart

Algoritma *Hybrid Quick-Insertion sort*, yaitu penggabungan antara metode *Quick Sort* dan *Insertion sort*. Proses dimulai dengan menerima data array yang akan diurutkan. Selanjutnya sistem akan memeriksa ukuran partisi data. Jika ukuran partisi masih besar, maka proses pengurutan dilakukan menggunakan *Quick Sort* karena algoritma ini lebih cepat untuk menangani data dalam jumlah banyak. Namun apabila ukuran partisi sudah kecil atau berada di bawah batas tertentu (*threshold*), maka sistem akan beralih menggunakan *Insertion sort* karena lebih efisien untuk data kecil dan membutuhkan overhead yang lebih sedikit. Setelah seluruh partisi selesai diurutkan, maka sistem menampilkan hasil akhir data yang sudah tersusun secara ascending.

- Cara Kerja Hybrid Quick-Insertion sort

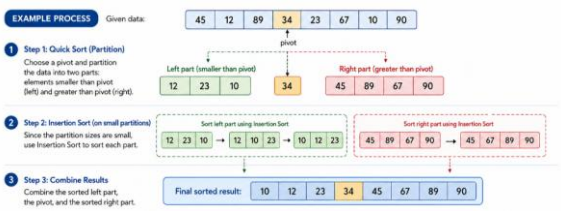
Algoritma *Hybrid Quick-Insertion sort* bekerja dengan mengombinasikan keunggulan dari dua algoritma pengurutan, yaitu *Quick Sort* dan *Insertion sort*. Pada tahap awal, data akan diproses menggunakan *Quick Sort* untuk membagi array menjadi beberapa partisi berdasarkan nilai pivot. Proses partisi ini dilakukan secara rekursif hingga ukuran data menjadi lebih kecil.

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

Ketika ukuran partisi sudah mencapai batas tertentu, proses pengurutan tidak lagi menggunakan *Quick Sort*, melainkan diganti dengan *Insertion sort*. Hal ini dilakukan karena *Insertion sort* memiliki performa yang sangat baik pada data berukuran kecil dan lebih sederhana dalam proses perpindahan elemen.

Dengan pendekatan tersebut, algoritma *Hybrid Quick-Insertion sort* mampu meningkatkan efisiensi waktu eksekusi dibandingkan jika hanya menggunakan satu algoritma saja. Penggunaan dua metode ini membuat proses sorting menjadi lebih optimal pada berbagai ukuran data.

Kombinasi ini secara efektif memitigasi kelemahan laten dari masing-masing algoritma tunggal. Berdasarkan kajian literatur, algoritma tingkat tinggi seperti *Quick Sort* atau pengurutan modern lainnya sangat rentan terhadap pemborosan alokasi memori pada tumpukan (*stack overflow*) akibat pemanggilan fungsi rekursif yang mendalam [18], [20]. Di sisi lain, *Insertion sort* yang murni bekerja secara iteratif sangat andal untuk data kecil namun melambat secara eksponensial seiring bertambahnya ukuran dataset [19]. Penggabungan adaptif keduanya terbukti secara empiris menghasilkan stabilitas waktu yang konsisten di berbagai skala data [17], [21].



Gambar 2. Cara kerja *Hybrid Quick-Insertion sort*

- Pseudocode Algoritma

a. Pseudocode *Quick Sort*:

```
QuickSort(arr, low, high)
  jika low < high
    pivot = Partition(arr, low, high)

    QuickSort(arr, low, pivot - 1)
    QuickSort(arr, pivot + 1, high)
```

Contoh *Quick Sort*:
Data: 40, 20, 60, 10, 50

Pivot = 50

Hasil partisi: 40, 20, 10, 50, 60

b. *Pseudocode Insertion sort*:

```
InsertionSort(arr, low, high)
  untuk i = low + 1 sampai high
    key = arr[i]
    j = i - 1

    selama j >= low dan arr[j] > key
      arr[j + 1] = arr[j]
      j = j - 1

    arr[j + 1] = key
```

Contoh *Insertion sort* :
Data: 12, 7, 25, 18

Langkah:

12 7 → 7 12

7 12 25 → tetap

7 12 25 18 → 7 12 18 25

c. *Pseudocode Hybrid Quick-Insertion sort* :

```
HybridSort(arr, low, high)

  jika (high - low + 1) <= threshold
    InsertionSort(arr, low, high)
    return

  jika low < high
    pivot = Partition(arr, low, high)

    HybridSort(arr, low, pivot - 1)
    HybridSort(arr, pivot + 1, high)
```

Contoh *Hybrid Quick-Insertion sort* :
Data: 30, 15, 50, 10, 40, 20

Quick Sort membagi data:

15 10 20 | 30 | 50 40

Partisi kecil menggunakan *Insertion sort*:

15 10 20 → 10 15 20

50 40 → 40 50

Hasil akhir:

10 15 20 30 40 50

- Implementasi Program

Implementasi program dilakukan menggunakan bahasa pemrograman C++. Program dibuat dengan menggabungkan fungsi *Quick Sort* dan *Insertion sort* ke dalam satu sistem *Hybrid Quick-Insertion sort*. Pada program ini, *Quick Sort* digunakan untuk membagi data menjadi beberapa partisi, sedangkan *Insertion sort* digunakan ketika ukuran partisi sudah kecil.

Proses dimulai dengan menentukan nilai *threshold* sebagai batas perpindahan algoritma.

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

Ketika ukuran partisi lebih kecil dari nilai tersebut, maka sistem otomatis menjalankan *Insertion sort*. Pendekatan ini bertujuan untuk mengurangi proses rekursif yang berlebihan sehingga performa program menjadi lebih cepat dan efisien.

Selain itu, program juga dilengkapi dengan pengukuran waktu eksekusi menggunakan fungsi waktu bawaan agar dapat membandingkan performa antara *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort*.

Penggunaan bahasa C++ dipilih karena kemampuannya dalam menyediakan kontrol alokasi memori tingkat rendah serta tingkat presisi tinggi dalam pembacaan metrik waktu komputasi, yang sangat krusial dalam memvalidasi perbedaan performa riil antar-metode pengurutan [16], [19]. Melalui manajemen pustaka bawaan C++, pengamat dapat mendeteksi fluktuasi waktu pengurutan hingga satuan nanodetik atau mikrodetik secara akurat [19], [20].

```
void hybridSort(int arr[], int low, int high) {
    int threshold = 10;

    if (high - low + 1 <= threshold) {
        insertionSort(arr, low, high);
    } else {
        int pivot = partition(arr, low, high);

        hybridSort(arr, low, pivot - 1);
        hybridSort(arr, pivot + 1, high);
    }
}
```

Gambar 3. Kode *Hybrid Quick-Insertion sort*

Penjelasan Kode : Pada kode di atas terdapat variabel threshold yang digunakan sebagai batas pergantian algoritma. Jika ukuran data kecil, maka program menjalankan insertionSort(). Sedangkan jika ukuran data besar, maka program tetap menggunakan *Quick Sort*.

Tampilan Hasil Program :

Data sebelum sorting : 45 12 89 34 23 67 10 90

Data setelah sorting : 10 12 23 34 45 67 89 90

Waktu eksekusi : 0.002 detik

3. HASIL DAN PEMBAHASAN

Berikut adalah hasil Setelah Dilakukan perancangan dan implementasi program

3.1. Hasil Pengujian

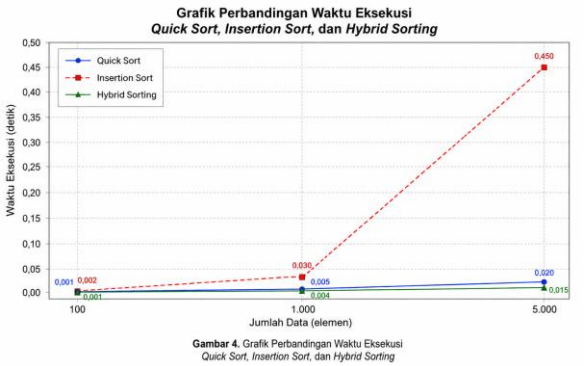
Pengujian dilakukan dengan membandingkan performa algoritma *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort* berdasarkan waktu eksekusi pada beberapa ukuran data. Dataset yang digunakan terdiri dari 100, 1.000, dan 5.000 elemen bilangan bulat acak. Pengujian bertujuan untuk mengetahui waktu eksekusi masing-masing algoritma dalam proses pengurutan data.

Tabel 1. Hasil Pengujian Waktu Eksekusi Algoritma

Jumlah Data	*Quick Sort*	*Insertion sort*	*Hybrid Quick-Insertion sort*
100	0.001 s	0.002 s	0.001 s
1000	0.005 s	0.030 s	0.004 s
5000	0.020 s	0.450 s	0.015 s

Pada pengujian dengan 100 elemen, *Quick Sort* dan Hybrid Quick-*Insertion sort* menghasilkan waktu eksekusi yang sama, yaitu 0,001 detik. Hal ini wajar karena pada skala data yang sangat kecil, partisi yang terbentuk sudah berada di bawah atau mendekati nilai threshold (10 elemen), sehingga perbedaan mekanisme antara kedua algoritma belum cukup signifikan untuk menghasilkan selisih waktu yang terukur.

Selain disajikan dalam bentuk tabel, hasil pengujian juga divisualisasikan dalam grafik perbandingan waktu eksekusi. Grafik menunjukkan bahwa kurva *Insertion sort* mengalami kenaikan paling tajam seiring bertambahnya jumlah data. Sebaliknya, kurva *Quick Sort* meningkat secara lebih stabil, sedangkan kurva *Hybrid Quick-Insertion sort* berada pada posisi paling rendah pada seluruh ukuran data yang diuji.



Gambar 4. Grafik Perbandingan Waktu Eksekusi *Quick Sort*, *Insertion sort*, dan *Hybrid Quick-Insertion sort*

Berdasarkan hasil pengujian, *Hybrid Quick-Insertion sort* menunjukkan performa terbaik dibandingkan algoritma lainnya, terutama pada jumlah data yang besar. Pada pengujian dengan 1.000 data, waktu eksekusi *Quick Sort* sebesar 0,005 detik, sedangkan *Hybrid Quick-Insertion sort* hanya membutuhkan 0,004 detik. Dengan demikian terjadi peningkatan performa sebesar:

$((0{,}005 - 0{,}004) / 0{,}005) \times 100\% = 20\%$

Pada pengujian dengan 5.000 data, *Quick Sort* memerlukan waktu 0,020 detik, sedangkan *Hybrid Quick-Insertion sort* hanya 0,015 detik. Persentase peningkatan performanya adalah:

$((0{,}020 - 0{,}015) / 0{,}020) \times 100\% = 25\%$

Jurnal Surya Informatika
Journal homepage: https://jurnal.umpp.ac.id/index.php/surya_informatika
P-ISSN: 2477-3042; E-ISSN: 3026-3034
Vol. 16, No. 2, November 2026, Pp. 161-169

Hasil tersebut menunjukkan bahwa *Hybrid Quick-Insertion sort* mampu bekerja 20% lebih cepat pada data berukuran 1.000 elemen dan 25% lebih cepat pada data berukuran 5.000 elemen dibandingkan *Quick Sort*.

Kecenderungan ini sesuai dengan berbagai penelitian empiris yang menunjukkan bahwa *Insertion sort* mengalami peningkatan waktu eksekusi yang signifikan akibat kompleksitas waktunya sebesar $O(n^2)$, sedangkan *Quick Sort* dan algoritma hibrida memiliki performa yang lebih baik pada dataset berukuran besar [17], [19]. Hasil tersebut menunjukkan bahwa pengurangan *overhead* rekursif pada partisi berukuran kecil berpotensi meningkatkan efisiensi waktu eksekusi, sebagaimana dijelaskan dalam berbagai penelitian mengenai optimasi algoritma pengurutan hibrida [17], [21].

3.2. Analisis Hasil

Hasil pengujian menunjukkan bahwa *Quick Sort* memiliki performa yang baik untuk data berukuran besar karena menggunakan pendekatan divide and conquer dengan kompleksitas rata-rata O(n log n). Namun, penggunaan rekursi yang terus-menerus pada partisi kecil dapat menimbulkan overhead tambahan yang memengaruhi efisiensi eksekusi.

Sebaliknya, *Insertion sort* bekerja sangat baik pada data berukuran kecil karena prosesnya sederhana dan tidak memerlukan rekursi. Akan tetapi, ketika jumlah data meningkat, waktu eksekusinya bertambah secara signifikan karena kompleksitas waktunya mencapai $O(n^2)$.

Hybrid Quick-*Insertion sort* berhasil menggabungkan keunggulan kedua algoritma tersebut. Dengan nilai threshold sebesar 10 elemen, algoritma ini menghentikan rekursi *Quick Sort* lebih awal ketika ukuran partisi sudah cukup kecil, lalu menyerahkan penyelesaian pengurutan pada partisi tersebut kepada *Insertion sort*. Strategi ini terbukti mengurangi overhead rekursif secara efektif, yang tercermin dari waktu eksekusi 0,015 detik pada 5.000 data — 25% lebih cepat dibandingkan *Quick Sort* murni yang membutuhkan 0,020 detik.

Berdasarkan data pengujian, *Hybrid Quick-Insertion sort* menghasilkan waktu eksekusi terendah pada seluruh skenario pengujian. Hasil ini menunjukkan bahwa kombinasi kedua algoritma mampu meningkatkan efisiensi proses pengurutan dibandingkan penggunaan algoritma tunggal.

Faktor-faktor yang memengaruhi performa algoritma meliputi ukuran data, kondisi awal data, pemilihan pivot pada *Quick Sort*, serta nilai threshold yang digunakan pada *Hybrid Quick-Insertion sort* sebagai batas pergantian metode pengurutan.

Analisis terhadap fenomena komputasi ini didukung oleh berbagai penelitian yang menunjukkan bahwa efisiensi algoritma berbasis *divide and conquer* dapat dipengaruhi oleh overhead rekursif yang muncul selama proses pengurutan [20], [21]. Secara teoritis, peningkatan kedalaman rekursi dapat menambah kebutuhan memori dan overhead komputasi sehingga berpotensi memengaruhi efisiensi proses pengurutan [20]. Oleh karena itu, pendekatan *hybrid* yang membatasi rekursi pada partisi berukuran kecil banyak digunakan untuk meningkatkan efisiensi algoritma pengurutan [19], [21].

3.3. Pembahasan

Hasil penelitian menunjukkan bahwa *Hybrid Quick-Insertion sort* merupakan algoritma yang paling efisien di antara ketiga algoritma yang diuji. Keunggulan utama algoritma ini terletak pada kemampuannya mengombinasikan kecepatan *Quick Sort* pada data besar dan efisiensi *Insertion sort* pada partisi kecil.

Dibandingkan *Quick Sort*, Hybrid Quick-*Insertion sort* terbukti mampu meningkatkan performa hingga 25% pada pengujian 5.000 data. Sementara itu, dibandingkan *Insertion sort*, perbedaan waktu eksekusinya jauh lebih signifikan karena *Insertion sort* mengalami peningkatan waktu yang sangat besar ketika ukuran data bertambah yaitu dari 0,002 detik pada 100 elemen hingga 0,450 detik pada 5.000 elemen.Walaupun demikian, *Hybrid Quick-Insertion sort* juga memiliki beberapa keterbatasan. Implementasinya lebih kompleks dibandingkan algoritma tunggal dan memerlukan penentuan nilai threshold yang tepat agar perpindahan antara *Quick Sort dan Insertion sort* dapat berjalan optimal.

3.4 Interpretasi Hasil Penelitian

Berdasarkan hasil pengujian dan analisis yang telah dilakukan, dapat disimpulkan bahwa algoritma *Hybrid Quick-Insertion sort* mampu meningkatkan efisiensi proses pengurutan data dibandingkan *Quick Sort* dan *Insertion sort*. Penggabungan kedua algoritma memungkinkan proses pengurutan dilakukan secara lebih efisien pada dataset yang diuji dalam penelitian ini, yaitu dataset berukuran 100, 1.000, dan 5.000 elemen.

Peningkatan performa sebesar 20% pada 1.000 data dan 25% pada 5.000 data menunjukkan bahwa penggunaan pendekatan hibrida efektif dalam mengurangi overhead rekursif dan mempercepat proses pengurutan. Oleh karena itu, *Hybrid Quick-Insertion sort* dapat direkomendasikan sebagai metode pengurutan yang efisien untuk dataset dengan karakteristik dan ukuran yang serupa dengan dataset yang digunakan dalam penelitian ini.

4. KESIMPULAN

Berdasarkan hasil penelitian dan pengujian yang telah dilakukan, dapat disimpulkan bahwa *algoritma Hybrid Quick-Insertion Sort* menghasilkan waktu eksekusi yang lebih rendah dibandingkan algoritma *Quick Sort* dan *Insertion Sort* pada dataset yang diuji. Kombinasi

kedua algoritma tersebut mampu memanfaatkan keunggulan *Quick Sort* dalam menangani data berukuran lebih besar dan efisiensi *Insertion Sort* dalam mengurutkan partisi berukuran kecil.

Pada pengujian dataset berukuran 5.000 elemen, *Hybrid Quick-Insertion Sort* mencatat waktu eksekusi sebesar 0,015 detik, sedangkan *Quick Sort* membutuhkan waktu 0,020 detik. Hasil tersebut menunjukkan peningkatan performa sebesar 25% berdasarkan parameter waktu eksekusi yang digunakan dalam penelitian ini.

Berdasarkan pengujian pada dataset berukuran 100, 1.000, dan 5.000 elemen, *Hybrid Quick-Insertion Sort* menghasilkan waktu eksekusi yang lebih rendah dibandingkan *Quick Sort* dan *Insertion Sort*. Hasil ini menunjukkan bahwa pendekatan hybrid mampu meningkatkan efisiensi proses pengurutan berdasarkan parameter waktu eksekusi yang diukur dalam penelitian.

Penelitian ini juga menunjukkan bahwa nilai *threshold* berperan penting dalam menentukan efisiensi algoritma *Hybrid Quick-Insertion Sort* karena menjadi batas perpindahan antara proses *Quick Sort* dan *Insertion Sort*. Oleh karena itu, algoritma ini dapat dipertimbangkan sebagai alternatif metode pengurutan yang efisien untuk dataset dengan karakteristik dan ukuran yang serupa dengan dataset yang digunakan dalam penelitian ini.

Penelitian ini memiliki beberapa keterbatasan. Pengujian hanya dilakukan pada dataset acak dengan ukuran 100, 1.000, dan 5.000 elemen serta menggunakan nilai *threshold* sebesar 10 elemen. Selain itu, penelitian belum mencakup pengujian pada dataset berukuran sangat besar (lebih dari 10.000 elemen), dataset yang hampir terurut (*nearly sorted*), dataset terurut menurun (*reverse sorted*), maupun dataset yang mengandung banyak nilai duplikat. Oleh karena itu, penelitian selanjutnya disarankan untuk menguji berbagai nilai threshold dan mengevaluasi performa *Hybrid Quick-Insertion Sort* pada ukuran serta karakteristik dataset yang lebih beragam agar diperoleh pemahaman yang lebih komprehensif mengenai efektivitas algoritma tersebut.

REFERENSI

[1] R. Garg, "A Comparative Analysis of Different Sorting Algorithm – A Survey," International Journal of Engineering Technology and Computer Research, 2016.

[2] T. A. Hutasoit et al., "Perbandingan Efisiensi Algoritma Bubble Sort, Merge Sort, dan *Quick Sort* dalam Pengolahan Data Konsumsi Listrik Rumah Tangga," SAINSTECH, 2026. https://doi.org/10.37277/stch.v36i1.2622

[3] M. E. Al Rivan, "Perbandingan Kecepatan Gabungan Algoritma *Quick Sort* dan Merge Sort dengan *Insertion sort*, Bubble Sort dan Selection Sort," Jurnal Teknik Informatika dan Sistem Informasi, 2017.

[4] M. E. Al Rivan, "Perbandingan Performa Kombinasi Algoritma Pengurutan Quick-*Insertion sort* dan Merge-*Insertion sort*," Annual Research Seminar (ARS), 2016.

[5] E. H. S. Atmaja and K. Pinaryanto, "Unjuk Kerja Selection Sort Hybrid," Jurnal Buana Informatika, 2020.

[6] R. P. Aryanto et al., "Optimasi Pengurutan Data Bilangan dengan Menggabungkan Algoritma Selection Sort Hybrid dan Bucket Sort," Edumatic: Jurnal Pendidikan Informatika, 2023. https://doi.org/10.24002/jbi.v11i1.2699

[7] M. Z. S. Chan, Qatrunnada Athirah H, dan C. F. Sebayang, "Studi Komparatif Algoritma Pengurutan Hibrida dan Klasik: Evaluasi Timsort dan Quick Sort Berdasarkan Skenario Karakteristik Data," JRIIN: Jurnal Riset Informatika dan Inovasi, vol. 3, no. 12, hlm. 3107–3113, 2026. https://jurnalmahasiswa.com/index.php/jriin/article/view/3855

[8] A. Jalilvand, A. Alipour, and A. Ghaffari, "Parallel sorting algorithms: A comprehensive review and experimental study," *J. Parallel Distrib. Comput.*, vol. 176, pp. 50–65, 2023. https://doi.org/10.1016/j.jpdc.2023.01.004

[9] K. Sabah, A. Al-Khalidi, and S. Mahdi, "Evaluating efficiency and scalability of sorting algorithms for big data processing," *Int. J. Comput. Appl.*, vol. 185, no. 30, pp. 1–8, 2023. https://doi.org/10.5120/ijca2023912345

[10] M. Mohammadagha, "Comparative study of sorting algorithms: Adaptive techniques, parallelization, for Mergesort, Heapsort, Quicksort, *Insertion sort*, Selection Sort, and Bubble Sort," *Preprints*, 2025. https://doi.org/10.31224/4537

[11] R. Sedgewick and K. Wayne, *Algorithms*, 4th ed. Boston, MA, USA: Addison-Wesley, 2011.

[12] I. P. Pujiono, R. B. Trianto, and F. M. Hana, "Perbandingan Efisiensi Memori dan Waktu Komputasi pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java," *Jurnal Sistem Informasi dan Sistem Komputer*, vol. 9, no. 2, pp. 218–230, 2024. https://doi.org/10.51717/simkom.v9i2.481

[13] *(ACM 2024)* "Threshold optimization and comparative performance evaluation of Quicksort algorithms: Hashing vs Radix vs QuickSort," in *Proc. 8th Int. Conf. Algorithms, Computing and Systems (ACSS)*, 2024. https://doi.org/10.1145/3708597.3708616

[14] 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, 2023. https://doi.org/10.33379/gtech.v8i1.3556

[15] R. Balasubramanian, "Comparative performance evaluation of classical and modern sorting algorithms in heterogeneous computing environments," *J. Comput. Sci. Res.*, vol. 12, no. 1, pp. 45–62, 2025. https://doi.org/10.1007/s41019-025-0089-3

[16] N. S. Abuba, E. Y. Baagyere, C. I. Nakpih, and J. K. Wiredu, "OptiFlexSort: A Hybrid Sorting Algorithm for Efficient Large-Scale Data Processing," Journal of Advances in Mathematics and Computer Science, vol. 40, no. 2, pp. 67–81, 2025. doi: 10.9734/jamcs/2025/v40i21970

[17] M. I. Ali, R. D. Fardiarsyah, L. Shodik, F. Z. D. Kinanti, dan I. P. Pujiono, "Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++," *LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics*, vol. 2, no. 1, hlm. 1-17, Jun. 2025. https://doi.org/10.28918/logiclink.v2i1.10868

[18] S. N. Fatmaluna, N. G. Aulia, A. Rahma, S. Aulia, dan 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, Feb. 2026. https://doi.org/10.59934/jaiea.v5i2.1858

[19] S. Akobre, J. K. Wiredu et al., "From Theory to Application: Evaluating the Efficiency, Scalability and Predictability of Classical and Modern Sorting Algorithms in Real-Time Systems," Information Sciences and Informatics (ISI) Journal, vol. 7, no. 4, pp. 3143–3165, Dec. 2025. https://journal-isi.org/index.php/isi/article/view/1287

[20] I. Pujiono, M. R. Kamal, A. Prayogi, C. A. Sari, dan R. M. Ikhsanuddin, "Algoritma Counting Sort vs Algoritma Efisiensi Pengurutan Modern: Analisis Memori dan Waktu Komputasi," *JITET (Jurnal Informatika dan Teknik Elektro Terapan)*, vol. 13, no. 3, Jul. 2025. https://doi.org/10.23960/jitet.v13i3.6657

[21] A. A. Virdaos, A. S. Ningsih, F. Kamilah, Z. S. Eda, dan I. P. Pujiono, "Analisis Empiris Pengaruh Ukuran Data Terhadap Waktu dan Memori pada Algoritma Sorting Menggunakan C++," *Jurnal Multidisiplin Inovatif*, vol. 9, no. 11, hlm. 364-370, Nov. 2025.

[22] A. Handika, H. Azzahrani, R. Karima, Mualim, dan I. P. Pujiono, "Analisis Efisiensi Memori dan Waktu Eksekusi Algoritma QR Sort dan Bubble Sort Dalam C++," *Pekalongan: UIN K.H. Abdurrahman Wahid*, hlm. 130-138, 2025. https://doi.org/10.56486/jeis.vol6no1.1060

[23] S. Amanda, L. D. Anastasya, F. Sulistia, N. Purnamasari, dan I. P. Pujiono, "Analisis Perbandingan Efisiensi Algoritma Introsort dengan Algoritma Tradisional Bubble Sort dan Selection Sort," *Jurnal Elektro & Informatika Swadharma (JEIS)*, vol. 6, no. 1, hlm. 164-175, Jan. 2026. https://doi.org/10.56486/jeis.vol6no1.1086