ANALISA EFISIENSI WAKTU KOMPUTASI DAN PENGGUNAAN MEMORI PADA ENAM ALGORITMA PENGURUTAN

Nafisha Putri Arsita[1)], Rahma Syarifa[2)], Tsaqib Fahmi Ahmad[3)], Imam Prayogo Pujiono[4)]
[1,2,3]Bisnis Digital, Fakultas Ekonomi dan Bisnis Islam, UIN K.H Abdurrahman Wahid Pekalongan
[4]Informatika, Fakultas Ekonomi dan Bisnis Islam, UIN K.H Abdurrahman Wahid Pekalongan

Correspondence author: NP Arsita, nafisha.putri.arsita.25012@mhs.uingusdur.ac.id,
Pekalongan, Indonesia

Abstract

This study aims to analyze and compare the performance of six sorting algorithms (Tim Sort, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Merge Sort) in terms of computational time efficiency and memory usage. Implementation was carried out in Java using Visual Studio Code as the development environment, and testing was conducted on a device with an Intel Core i7-8665U (1.02 GHz) processor, 8 GB of RAM, a 64-bit Operating System, and a 256 GB SSD. All algorithms were tested using three different dataset sizes, namely 100, 1,000, and 5,000 data points, each with three repetitions to obtain more accurate results. Based on the test, Tim Sort showed the most efficient performance, maintaining stable computational time and memory usage across all dataset sizes. Quick Sort and Merge Sort have also been shown to perform well, especially for large data, in contrast to Bubble Sort, Selection Sort, and Insertion Sort, which become less efficient as data grows due to significantly increased processing time. Overall, this study concludes that Tim Sort is the most optimal algorithm in terms of speed and memory efficiency. Quick Sort and Merge Sort are recommended for sorting large-scale data. At the same time, basic algorithms like Bubble Sort, Selection Sort, and Insertion Sort are better suited to small datasets or to learning sorting concepts.

Keyword *: comparison, sorting algorithms, computation time, memory usage*

Abstrak

Penelitian ini bertujuan untuk manganalisis dan membandingkan performa enam algortima *sorting (Tim Sort, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, dan Merge Sort*) dari aspek efisiensi waktu komputasi dan penggunaan memori. Implementasi dilakukan menggunakan Bahasa pemrograman Java melalui *Visual Studio Code* sebagai lingkungan pengembangan, dan pengujian dijalankan pada perangkat dengan spesifikasi *Intel Core* i7-8665U (1,02 GHz), RAM 8 GB, Sistem Operasi 64-bit, serta SSD 256 GB. Semua algoritma diuji menggunakan tiga perbedaan ukuran dataset, yaitu 100 data, 1.000 data, dan 5.000 data, masing-masing dengan tiga kali pengulangan untuk mendapatkan hasil yang lebih akurat. Berdasarkan pengujian ditemukan bahwa *Tim Sort* menunjukkan kinerja yang paling efisien, mempertahankan waktu komputasi, dan pemakaian memori yang stabil pada semua ukuran dataset. *Quick Sort* dan *Merge Sort* juga terbukti memiliki performa kuat, khususnya ketika menangani data berukuran

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

besar, berbeda dengan *Bubble Sort, Selection Sort,* dan *Insertion Sort* yang menjadi kurang efisien karena peningkatan waktu proses yang signifikan seiring pertumbuhan data. Secara keseluruhan, penelitian ini menyimpulkan bahwa *Tim Sort* merupakan algoritma yang paling optimal dalam aspek kecepatan dan efisiensi memori. Adapun *Quick Sort* dan *Merge Sort* direkomendasikan untuk pengurutan data dalam skala besar, sedangkan algoritma dasar seperti *Bubble Sort, Selection Sort,* dan *Insertion Sort* lebih sesuai digunakan untuk dataset kecil atau keperluan pembelajaran konsep pengurutan.

Kata Kunci : perbandingan, algoritma pengurutan, waktu komputasi, penggunaan memori

A. PENDAHULUAN

Pengurutan data (*Sorting*) merupakan operasi fundamental dalam menentukan seberapa efisiens sistem dapat mengelola dan menyajikan data (Ghofur et al., 2025). Jumlah data yang diproses memiliki dampak langsung terhadap kinerja sistem Selama proses pengurutan (Sari et al., 2022). Dalam Bahasa pemrograman Java, terdapat berbagai algoritma pengurutan seperti *Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, dan Merge Sort.* Setiap algoritma tersebut memiliki karakteristik berbeda dalam hal efisiensi memori dan waktu eksekusi (Sena et al., 2024).

Efisiensi memori dan waktu komputasi menjadi faktor utama dalam menentukan kinerja suatu algoritma pengurutan data (Basir, 2020). Pemilihan algoritma yang tepat dapat memimalkan penggunaan sumber daya memori dan waktu pemrosesan yang berlebihan (Pujiono et al., 2025). Seiring dengan perkembangan teknologi komputasi, berbagai algoritma pengurutan baru telah dikembangkan dengan tujuan meningkatkan efisiensi dan kecepatan dlam pengolahan dataset berukuran besar. Salah satu agoritma pengurutan modern yang banyak digunakan saat ini adalah *Tim Sort*.

Tim Sort merupakan algoritma hibdrida yang menggabungkan keunggulan dari *Insertion Sort* dan *Merge Sort,* sehingga mampu menghasilkan kinerja yang efisien, cepat, dan stabil pada berbagai kondisi data (Al-aydi & Abu-naser, 2025). Penelitian (Hanafi et al., 2022) menjelaskan bahwa langkah-langkah umum dalam algoritma *Tim Sort* meliputi : membagi array menjadi beberapa blok yang disebut *run*, menentukan ukuran *run,* umumnya 32 atau 64 elemen, kemudian mengurutkan elemen pada setiap *run* menggunakan *Insertion Sort*, dan terakhir menggabungkan setiap *run* yang telah diurutkan dengan metode *Merge Sort.*

Berbagi penelitian sebelumnya telah mengkaji efisiensi memori dan waktu komputasi pada algoritma pengurutan data yang diimplementasikan dalam Bahasa pemrograman Java sebagai upaya untuk meningkatkan kinerja komputasi. Penelitian (M. I. Ali et al., 2025) menganalisis efisiensi mempori dan waktu komputasi pada delapan algoritma *sorting* menggunakan Bahasa C++. Penelitian lainnya (H. Ali et al., 2021) membandingkan performa antara algoritma *Heap Sort* dan *Insertion Sort.* Selanjutnya penelitian (Mahrozi & Faisal, 2023) yang dilakukan pada tahun 2023 menganalisis perbandingan kecepatan dari segi waktu antara algoritma pengurutan data *Selection Sort* dan *Bubble Sort.* Penelitian lain (Iskandar et al., 2020) membahas analisis efisiensi memori pada algoritma *Bubble Sort* dan *Insertion Sort.* Sementara itu, penelitian yang dilakukan pada bulan Juli tahun 2025 (Musyaffa et al., 2025) mengkaji efisiensi memori dan waktu pada proses pengurutan data menggunakan algoritma ASA dan algoritma pengurutan tradisional berbasis

Python. Selain itu, penelitian (Isyqi et al., 2024) melakukan perbandingan kinerja antara algoritma *Bubble Sort, Insertion Sort,* dan *Selection Sort* menggunakan data numerik dalam implementasi bahasa Python. Riset tahun 2025 (Ilham et al., 2025) melakukan analisis komparatif terhadap lima algoritma pengurutan untuk menilai kecepatan pemrosesan serta efisiensi penggunaan memori ketika diimplementasikan menggunakan Bahasa pemrograman C++. Pada kajian lain (Gudiato et al., 2024), dilakukan analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python. Sementara itu, penelitian sebelumnya (Latifah et al., 2021) mengkaji perbandingan kinerja antara algoritma Bubble Sort dan Selection Sort dalam konteks pemrograman paralel. Penelitian lain (Wijaya et al., 2024) menyoroti perbandingan algoritma pengurutan menggunakan Bahasa pemrograman JavaScript dengan fokus pada aspek waktu komputasi dan penggunaan memori. Selanjutnya, penelitian tahun 2025 (Sutanto et al., 2025) membahas efisiensi waktu pengurutan data antara *Bubble Sort, Insertion Sort, Merge Sort, Quick Sort* menggunakan bahasa Python. Pada penelitian berikutnya (Amanda et al., 2026), Evaluasi kinerja kombinasi algoritma sorting *Intro Sort, Bubble Sort* dan *Selection Sort.* Adapun penelitian tahun 2022 (Zulfa et al., 2022) membandingkan kinerja algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam proses pengurutan bilangan acak menggunakan Bahasa pemrograman Java.

Pada penelitian ini dilakukan analisis terhadap efisiensi memori dan waktu komputasi pada algoritma pengurutan data, dengan membandingkan algoritma Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort yang diimplementasikan menggunakan pemrograman Java. Data tersebut diuji dalam tiga variasi ukuran, yaitu 100, 1000, dan 5000 elemen. Setiap pengujian dilakukan sebanyak tiga kali dengan menggunakan aplikasi yang sama yaitu *Visual Studio Code* untuk memperoleh hasil yang lebih akurat. Pengukuran efisiensi memori dilakukan berdasarkan jumlah memori yang digunakan selama proses eksekusi, sedangkan waktu komputasi diukur dari durasi yang dibutuhkan komputer untuk menyelesaikan proses pengurutan data secara keseluruhan. Selanjutnya, kinerja algoritma *Tim Sort* dianalisis dan dikomparasikan dengan lima algoritma pengurutan lainya guna mengidentifikasi tingkat efisiensi *Tim Sort* dalam pemanfaatan memori serta waktu komputasi. Kontribusi utama penelitian ini adalah menyajikan analisis komparatif yang terukur antara *Tim Sort* dan algoritma pengurutan tradisional pada Bahasa pemrograman Java dengan dua perpsektif sekaligus yaitu efisiensi waktu dan efisiensi memori, serta merumuskan rekomendasi praktis pemilihan algoritma untuk berbagai skala data.

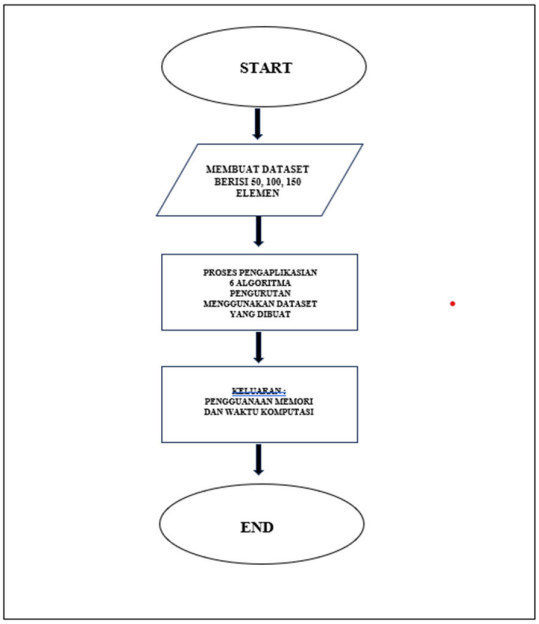
B. METODE PENELITIAN

Penelitian ini menggunakan metode komparatif dengan tujuan membandingkan tingkat efisiensi beberapa algoritma pengurutan data. Algoritma yang diuji meliputi *Tim Sort* sebagai algoritma modern serta beberapa algoritma pengurutan tradisional, yaitu *Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, dan Merge Sort.* Melalui pendekatan komparatif ini, penelitian bertujuan mengetahui algoritma yang paling efisien berdasarkan penggunaan waktu komputasi dan memori. Langkah pertama dalam penelitian ini diawali dengan pembuatan dataset yang digunakan sebagai data uji. Dataset tersebut dibagi ke dalam tiga kategori ukuran, masing-masing memiliki rentang nilai antara 1 hingga 99. Dataset berukuran kecil berisi 100 elemen, dataset berukuran sedang berisi 1.000 elemen, dan dataset berukuran besar berisi 5.000 elemen. Seluruh elemen data dihasilkan menggunakan aplikasi *Random Number Generator Plus*. Pada aplikasi tersebut,

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

peneliti menentukan nilai minimum, nilai maksimum, serta jumlah elemen yang ingin dihasilkan untuk memperoleh kumpulan data acak secara otomatis.

Dataset yang telah diperoleh kemudian diimplementasikan ke dalam kode program menggunakan bahasa pemrograman Java. Setiap algoritma pengurutan diuji secara terpisah dengan dataset yang sama untuk memastikan hasil pengukuran bersifat objektif. Pengujian dilakukan dengan mengukur waktu komputasi dan penggunaan memori pada setiap algoritma. Proses pengukuran dilakukan dengan memanfaatkan kelas Runtime dalam Java untuk memperoleh data terkait memori sebelum dan sesudah proses pengurutan. Hasil pengujian dari setiap algoritma kemudian dibandingkan dan dianalisis untuk menentukan algoritma yang paling efisien berdasarkan kedua parameter tersebut. Gambar 1 menunjukkan tahapan dari penelitian ini.



Gambar 1. Diagram Alir Tahapan Penelitian

C. HASIL DAN PEMBAHASAN

Sistem Penelitian ini memanfaatkan bahasa pemrograman Java dan *Integrated Development Environment* (IDE) *Visual Studio Code Versi 1.105.1* sebagai sarana untuk pengembangan sekaligus eksekusi kode program. Proses pengujian dijalankan menggunakan laptop dengan spesifikasi Intel Core i7-8665U dengan kecepatan 1,02 GHz, memori 8 GB RAM, sistem Operasi 64 Bit, dan SSD berkapasitas 256 GB.

Algoritma yang diuji dalam penelitian ini meliputi *Tim Sort, Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, dan Merge Sort*. Seluruh algoritma tersebut diuji menggunakan dataset yang sama, yaitu 100 data (array kecil), 1.000 data (array sedang), dan 5.000 data (array besar) menggunakan data numerik acak dengan rentang nilai 1 hingga 99.

Proses pengujian dilakukan sebanyak tiga kali untuk setiap ukuran dataset dengan rincian sebagai berikut:

1. Pengujian pertama hingga ketiga menggunakan dataset berisi 100 data (array kecil).
2. Pengujian keempat hingga keenam menggunakan dataset berisi 1.000 data (array sedang).
3. Pengujian ketujuh hingga kesembilan menggunakan dataset berisi 5.000 data (array besar).

Setiap algoritma diuji dan diulang selama tiga kali agar mendapat hasil yang tepat. Pengujian dilakukan menggunakan aplikasi *Visual Studio Code* sebagai lingkungan pengembangan untuk menjalankan program, sedangkan hasil pengujian dicatat menggunakan Microsoft Word.

Dataset lengkap pengujian dapat diakses melalui tautan bit.ly/4hVK923. Selama proses pengujian, setiap algoritma dievaluasi berdasarkan jumlah memori yang digunakan serta lama waktu yang diperlukan untuk menghasilkan urutan data. Berikut merupakan tautan hasil dokumentasi untuk masing-masing pengujian: link *Tim Sort*

(bit.ly/47EK9yI). Link *Bubble Sort* (bit.ly/3WF7rQ3). Link *Selection Sort* (bit.ly/4qKBNOD). Link *Insertion Sort* (bit.ly/3LjXhlw). Link *Quick Sort* (bit.ly/4p1QfAu). Dan link *Merge Sort* (bit.ly/4qNZqWy/).

Tim Sort

```
// === Fungsi utama TimSort ===
public static void timSort(int[] arr, int n) {
   int RUN = 32; // Ukuran blok kecil untuk Insertion Sort

   // 1. Urutkan setiap blok kecil menggunakan Insertion Sort
   for (int i = 0; i < n; i += RUN) {
      for (int j = i + 1; j < Math.min(i + RUN, n); j++) {
         int temp = arr[j];
         int k = j - 1;
         while (k >= i && arr[k] > temp) {
            arr[k + 1] = arr[k]; // Geser elemen lebih besar
            k--;
         }
         arr[k + 1] = temp; // Tempatkan elemen di posisi yang benar
      }
   }
   // 2. Gabungkan blok-blok yang sudah terurut secara bertahap
   for (int size = RUN; size < n; size *= 2) {
      for (int left = 0; left < n; left += 2 * size) {
         int mid = left + size - 1;
         int right = Math.min(left + 2 * size - 1, n - 1);
         if (mid < right) {
            // Merge dua blok terurut
            int[] leftArr = Arrays.copyOfRange(arr, left, mid + 1);
            int[] rightArr = Arrays.copyOfRange(arr, mid + 1, right + 1);
            int i = 0, j = 0, k = left;
            while (i < leftArr.length && j < rightArr.length) {
               arr[k++] = (leftArr[i] <= rightArr[j]) ? leftArr[i++] : rightArr[j++];
            }
            while (i < leftArr.length) arr[k++] = leftArr[i++];
            while (j < rightArr.length) arr[k++] = rightArr[j++];
         }
      }
   }
}
```

Kode program diatas merupakan implementasi dari algortima *Tim Sort*, Pola operasional algoritma *TimSort* merepresentasikan hibridisasi antara pendekatan *Insertion Sort* dan *Merge Sort*, yang dirancang untuk memaksimalkan efisiensi proses pengurutan. Algoritma ini diawali dengan pembagian struktur data array menjadi sejumlah segmen kecil yang dikenal sebagai *run*. Setiap *run* kemudian diurutkan secara individual menggunakan metode Insertion Sort, mengingat algoritma ini menunjukkan kinerja optimal untuk pengurutan data dalam skala terbatas. Setelah seluruh *run* berhasil tersusun secara terurut, tahap berikutnya melibatkan penggabungan (*merge*) *run*-run tersebut secara bertahap, mengikuti mekanisme yang serupa dengan algoritma *Merge Sort*. Proses penggabungan dilakukan secara sistematis hingga seluruh segmen array terintegrasi menjadi satu kesatuan yang tersusun secara konsisten. Pendekatan hibrida ini memungkinkan *TimSort* beroperasi secara optimal terutama ketika data awal sudah memiliki sebagian elemen yang tersusun, sekaligus mempertahankan stabilitas pengurutan, yakni urutan relatif elemen dengan nilai identik tetap terjaga. Pada akhirnya, algoritma ini menghasilkan array yang tersusun secara menaik (*ascending order*), mulai dari nilai terkecil hingga terbesar, dengan efisiensi signifikan baik dari sisi waktu proses maupun penggunaan memori.

Tabel 1. Hasil Pengujian Algoritma Tim Sort

Uji Ke	Hasil Pengujian Algoritma Tim Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	1.2786
2	100	40	3456
3	100	40	1.5763
4	1.000	40	4.4448
5	1.000	40	3.8101
6	1.000	40	5.7755
7	5.000	251	22.8013
8	5.000	251	15.1905
9	5.000	251	25.1534

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

Berdasarkan hasil pengujian tabel 1, dari perspektif memori, rata-rata konsumsi memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data adalah 40 KB, dan dengan 5.000 data adalah 251 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.40016 milidetik, dengan 1.000 data adalah 4.6768 milidetik dan dengan 5.000 data adalah 21.0484 milidetik.

Bubble Sort

```
int n = arr.length;
for (int i = 0; i < n - 1; i++) {        // Loop luar: setiap pass
   boolean swapped = false;              // Flag untuk cek pertukaran
   for (int j = 0; j < n - i - 1; j++) {  // Loop dalam: bandingkan elemen berurutan
      if (arr[j] > arr[j + 1]) {         // Jika tidak urut
         int temp = arr[j];              // Tukar posisi
         arr[j] = arr[j + 1];
         arr[j + 1] = temp;
         swapped = true;                 // Tandai ada pertukaran
      }
   }
   if (!swapped) break;                  // Hentikan jika array sudah terurut
}
```

Kode program diatas merupakan implementasi dari algortima *Bubble Sort*, Pola operasional algoritma *Bubble Sort* dilaksanakan melalui mekanisme perbandingan antar dua elemen yang berposisi bersebelahan dalam struktur data. Apabila elemen pada posisi kanan memiliki nilai numerik lebih kecil dibandingkan elemen pada posisi kiri, maka kedua elemen tersebut mengalami proses pertukaran (*swap*). Proses pertukaran ini dilakukan secara iteratif sepanjang satu siklus evaluasi, sehingga setiap pasangan elemen diperiksa secara menyeluruh. Setelah satu siklus selesai, algoritma melanjutkan ke siklus berikutnya dengan pola operasi yang identik, hingga tercapai kondisi di mana tidak terjadi pertukaran nilai sama sekali. Kondisi ini menandakan bahwa seluruh elemen telah tersusun secara benar sesuai urutan yang diinginkan. Dengan demikian, algoritma Bubble Sort mampu menghasilkan susunan data secara menaik (ascending order), dimulai dari nilai terkecil hingga nilai terbesar, melalui proses iteratif yang sistematis dan deterministik.

Tabel 2. Hasil Pengujian Algoritma Bubble Sort

Uji Ke	Hasil Pengujian Algoritma Bubble Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	4.5252
2	100	40	1.3303
3	100	40	2.0375
4	1.000	40	8.6362
5	1.000	40	9.8187
6	1.000	40	9.0292
7	5.000	96	56.4221
8	5.000	96	30.6994
9	5.000	96	40.2875

Berdasarkan hasil pengujian tabel 2, jika dilihat dari segi memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data adalah 40 KB, dan dengan 5.000 data adalah 96 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 2.631 milidetik, dengan 1.000 data adalah 9.16136 milidetik, dan dengan 5.000 data adalah 188,097.3289. milidetik.

Selection Sort

```
int n = arr.length;

for (int i = 0; i < n - 1; i++) {
   int minIndex = i;

   // Cari elemen terkecil di sisa array
   for (int j = i + 1; j < n; j++) {
      if (arr[j] < arr[minIndex]) {
         minIndex = j;
      }
   }

   // Tukar elemen terkecil dengan elemen pertama dari bagian belum terurut
   if (minIndex != i) {
      int temp = arr[minIndex];
```

```
            arr[minIndex] = arr[i];
            arr[i] = temp;
        }
    }
```

Kode program diatas merupakan implementasi dari algortima *Selection Sort*, Pola operasional algoritma *Selection Sort* dijalankan melalui mekanisme seleksi nilai terkecil dari keseluruhan elemen yang terdapat dalam array. Nilai terkecil yang teridentifikasi pada tahap awal kemudian ditukar (*swap*) dengan elemen yang berada pada posisi indeks pertama. Setelah pertukaran tersebut, proses pencarian nilai terkecil dilanjutkan pada subset elemen yang tersisa, dimulai dari indeks kedua, dan nilai terkecil yang ditemukan pada tahap ini ditukar dengan elemen pada posisi indeks kedua. Mekanisme pencarian dan pertukaran ini berlangsung secara iteratif hingga seluruh elemen menempati posisi yang sesuai sesuai urutan yang diinginkan. Dengan demikian, algoritma Selection Sort menghasilkan susunan data secara menaik (ascending order), dimulai dari nilai terkecil hingga nilai terbesar. Jumlah pertukaran yang terjadi bergantung secara langsung pada jumlah elemen dalam array, sedangkan proses pencarian nilai minimum memerlukan evaluasi berulang pada seluruh elemen yang tersisa pada setiap iterasi, yang memengaruhi kompleksitas waktu keseluruhan algoritma.

Tabel 3. Hasil Pengujian Algoritma Selection Sort

Uji Ke	Hasil Pengujian Algoritma Selection Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	1.869
2	100	40	1.1346
3	100	40	1.9142
4	1.000	40	8.6803
5	1.000	40	8.7331
6	1.000	40	8.4616
7	5.000	96	31.0111
8	5.000	96	33.2489
9	5.000	96	41.1057

Berdasarkan hasil pengujian tabel 3, jika dilihat dari segi memori, rata-rata penggunaan memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data ialah 40 KB, dan dengan 5.000 data ialah 96 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.63926 milidetik, dengan 1.000 data adalah 25.875 milidetik, dan dengan 5.000 data adalah 25.1219 milidetik.

Insertion Sort

```
int n = arr.length;

for (int i = 1; i < n; i++) {
    int key = arr[i];
    int j = i - 1;

    // Geser elemen yang lebih besar dari key ke kanan
    while (j >= 0 && arr[j] > key) {
        arr[j + 1] = arr[j];
        j--;
    }
    arr[j + 1] = key; // Tempatkan key di posisi yang
tepat
}
```

Kode program diatas merupakan implementasi dari algortima *Insertion Sort*, Pola operasional algoritma *Insertion Sort* dimulai dengan asumsi bahwa elemen pertama dalam array telah berada pada posisi yang benar, sementara elemen-elemen berikutnya masih dalam keadaan belum terurut. Proses pengurutan dimulai dari elemen kedua, yang dijadikan sebagai kunci (*key*). Elemen kunci ini kemudian dibandingkan secara berurutan dengan elemen-elemen di sebelah kirinya. Apabila elemen sebelah kiri memiliki nilai yang lebih besar daripada kunci, maka elemen tersebut digeser satu posisi ke arah kanan untuk memberikan ruang bagi penyisipan kunci. Pergeseran ini dilakukan secara iteratif hingga ditemukan posisi yang tepat, yakni posisi tempat sebelumnya memiliki nilai lebih kecil atau sama dengan kunci. Setelah posisi yang sesuai teridentifikasi, kunci disisipkan ke dalam posisi tersebut.

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

Prosedur ini berlanjut secara berurutan untuk elemen ketiga, keempat, dan seterusnya, sehingga pada setiap langkah, subarray di sebelah kiri elemen kunci selalu berada dalam keadaan terurut. Proses iteratif ini berakhir ketika seluruh elemen array telah melalui tahap perbandingan dan penyisipan, menghasilkan susunan data yang terurut secara menaik (*ascending order*), mulai dari nilai terkecil hingga nilai terbesar. Pendekatan ini memastikan stabilitas pengurutan, karena urutan relatif elemen dengan nilai identik tetap terjaga, sekaligus meminimalkan jumlah pergeseran yang diperlukan untuk data yang sebagian sudah terurut.

Tabel 4. Hasil Pengujian Algoritma *Insertion Sort*

Uji Ke	Hasil Pengujian Algoritma Insertion Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	1.9303
2	100	40	1.9466
3	100	40	1.0181
4	1.000	40	6.1726
5	1.000	40	7.8368
6	1.000	40	7.6174
7	5.000	96	22.2042
8	5.000	96	22.9072
9	5.000	96	28.2614

Berdasarkan hasil pengujian tabel , jika dilihat dari segi memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data adalah 40 KB, dan dengan 5.000 data adalah 96 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.6316 milidetik, dengan 1.000 data adalah 7.20893 milidetik, dan dengan 5.000 data adalah 24.4576 milidetik.

Quick Sort

```
if (low < high) {
int pivotIndex = partition(arr, low, high); // Tentukan posisi pivot
   quickSort(arr, low, pivotIndex - 1);      // Rekursif ke kiri pivot
   quickSort(arr, pivotIndex + 1, high);     // Rekursif ke kanan pivot
}
```

Kode program diatas merupakan implementasi dari algortima *Quick Sort*, Pola operasional algoritma *Quick Sort* dimulai dengan pemilihan satu elemen sebagai pivot, yang berfungsi sebagai acuan utama dalam proses pengurutan. Seluruh elemen lain dalam array dibandingkan dengan nilai acuan, di mana elemen dengan nilai lebih kecil dari nilai acuan ditempatkan pada sisi kiri, sedangkan elemen dengan nilai lebih besar ditempatkan pada sisi kanan. Proses pembagian ini dikenal sebagai partitioning, yang menjamin bahwa pivot menempati posisi finalnya yang benar dalam urutan akhir setelah setiap partisi.

Setelah tahap partitioning selesai, algoritma diterapkan kembali secara rekursif pada subarray kiri dan kanan, yakni pada kelompok elemen dengan nilai lebih kecil maupun lebih besar daripada pivot. Proses rekursif ini terus berlangsung hingga setiap subarray hanya terdiri dari satu elemen atau tidak lagi memerlukan pengurutan tambahan. Pendekatan ini memungkinkan pembentukan struktur data yang terurut secara bertahap melalui pembagian dan penggabungan yang sistematis.

Dengan demikian, algoritma Quick Sort menghasilkan susunan data menaik (ascending order), dari nilai terkecil hingga terbesar, dengan pivot berperan sentral dalam menentukan posisi elemen dan memfasilitasi efisiensi pengurutan. Metode ini memiliki keunggulan dalam hal kompleksitas waktu rata-rata, meskipun kinerjanya dapat dipengaruhi oleh pemilihan pivot pada dataset tertentu.

Tabel 5. Hasil Pengujian Algoritma Quick Sort

Uji Ke	Hasil Pengujian Algoritma Quick Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	2.4579
2	100	40	0.9396

Uji Ke	Hasil Pengujian Algoritma Quick Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
3	100	40	0.8901
4	1.000	40	4.7131
5	1.000	40	3.6
6	1.000	40	5.2718
7	5.000	96	13.7851
8	5.000	96	14.6089
9	5.000	96	16.9416

Berdasarkan hasil pengujian tabel 5, ditinjaudari prespektif konsumsi memori, rata-rata penggunaan memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data adalah 40 KB, dan dengan 5.000 data adalah 96 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.43183 milidetik, dengan 1.000 data adalah 13.5849 milidetik, dan dengan 5.000 data adalah 15.11186 milidetik.

Merge Sort

```
if (left < right) {
   int mid = (left + right) / 2;

   // Rekursi untuk bagian kiri dan kanan
   mergeSort(arr, left, mid);
   mergeSort(arr, mid + 1, right);

   // Gabungkan hasil pengurutan
   merge(arr, left, mid, right);
}
```

Kode program diatas merupakan implementasi dari algortima *Merge Sort*, Pola operasional algoritma *Merge Sort* dimulai dengan pembagian (divide) array secara rekursif menjadi dua subarray hingga setiap subarray hanya terdiri atas satu elemen, kondisi di mana setiap elemen secara implisit dianggap telah terurut. Tahap berikutnya adalah proses penggabungan (merge), di mana dua subarray yang lebih kecil digabung menjadi satu subarray yang lebih besar dengan mempertahankan keterurutan. Proses penggabungan dilakukan dengan membandingkan elemen-elemen dari kedua subarray dan menempatkannya secara berurutan sehingga membentuk urutan menaik (ascending order), mulai dari nilai terkecil hingga terbesar.

Prosedur pembagian dan penggabungan ini berlangsung secara iteratif dan rekursif hingga seluruh subarray kembali terintegrasi menjadi satu kesatuan array yang telah tersusun secara sempurna. Pendekatan ini merupakan implementasi dari strategi divide and conquer, yakni memecah masalah besar menjadi bagian-bagian yang lebih kecil, menyelesaikan setiap bagian secara independen, dan kemudian menggabungkannya kembali untuk memperoleh hasil pengurutan yang optimal. Metode ini menjamin stabilitas pengurutan, karena posisi relatif elemen dengan nilai identik tetap dipertahankan, serta efisiensi waktu yang konsisten terutama pada dataset dengan ukuran besar.

Tabel 6. Hasil Pengujian Algoritma Merge Sort

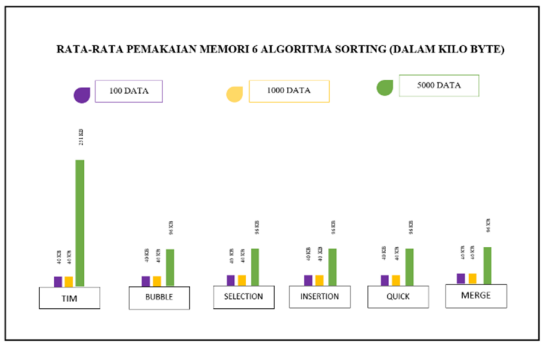
Uji Ke	Hasil Pengujian Algoritma Merge Sort		
	Jumlah Data	Pemakaian Memori (KB)	Waktu Komputasi (milidetik)
1	100	40	1.5774
2	100	40	0.9875
3	100	40	1.9832
4	1.000	40	33.0365
5	1.000	40	3.5722
6	1.000	40	5.0721
7	5.000	96	14.1419
8	5.000	96	18.155
9	5.000	96	14.4867

Berdasarkan hasil pengujian tabel 6, jika dilihat dari segi memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 40 KB, dengan 1.000 data adalah 122 KB, dan dengan 5.000 data adalah 531 KB. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.49773 milidetik, dengan 1.000 data adalah 13.8936 milidetik, dan dengan 5.000 data adalah 15.59453 milidetik.

Dari hasil pengujian keenam algoritma yang sudah dilakukan, rangkuman rata-rata

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

hasil pengujian setiap algoritma dapat dilihat pada gambar 2 dan 3 dibawah ini.



Gambar 2. Diagram Rata-rata Pemakaian Memori

Gambar 3. Diagram Rata-rata Kecepatan memori

Selection Sort, Insertion Sort, Quick Sort, dan Merge Sort, diperoleh perbedaan kinerja yang signifikan dalam hal waktu komputasi dan pemakaian memori pada berbagai ukuran dataset. Secara umum, *Tim Sort* menunjukkan performa paling efisien dan stabil dibandingkan algoritma lainnya. Hal ini disebabkan oleh sifat hibridanya yang menggabungkan keunggulan Insertion Sort untuk data berukuran kecil dan *Merge Sort* untuk proses penggabungan berskala besar, namun pada sisi penggunaan memori jika semakin banyak data yang dimasukkan maka semakin banyak memori yang digunakan.

Algoritma *Quick Sort* juga menunjukkan waktu komputasi yang kompetitif pada berbagai ukuran data, terutama karena penerapan strategi *divide and conquer* yang efektif dalam membagi dan mengurutkan data. Namun, efisiensinya sangat bergantung pada pemilihan pivot, yang dapat memengaruhi performa secara signifikan.

Sementara itu, *Merge Sort* memberikan hasil yang konsisten dan stabil dengan kompleksitas waktu yang baik pada dataset besar, meskipun memerlukan penggunaan memori tambahan yang relatif lebih tinggi dibandingkan algoritma lain. Berbeda dengan ketiga algoritma tersebut, algoritma pengurutan sederhana seperti *Bubble Sort*, *Selection Sort, dan Insertion Sort* menunjukkan performa yang kurang efisien terutama pada dataset berukuran besar. Ketiga algoritma ini memerlukan waktu komputasi yang jauh lebih lama akibat banyaknya proses perbandingan dan pertukaran elemen.

D. PENUTUP

Kesimpulan dari penelitian ini, Tim Sort merupakan algoritma paling efisien dalam hal waktu dan penggunaan memori. Quick Sort dan Merge Sort juga cepat untuk data besar, sedangkan Bubble Sort, Selection Sort, dan Insertion Sort jauh lebih lambat dan hanya cocok untuk dataset kecil atau pembelajaran dasar. Berdasarkan hasil penelitian ini, pembaca disarankan untuk mempertimbangkan penggunaan Tim Sort atau Quick Sort ketika menangani proses pengurutan pada dataset berukuran sedang hingga besar karena keduanya menunjukkan kinerja komputasi yang lebih efisien. Selain itu, Merge Sort dapat menjadi alternatif ketika dibutuhkan hasil pengurutan yang stabil meskipun memerlukan memori tambahan. Sebaliknya, penggunaan algoritma Bubble Sort, Selection Sort, dan Insertion Sort sebaiknya dihindari untuk dataset besar karena waktu komputasi yang kurang optimal. Pemilihan algoritma pengurutan sebaiknya disesuaikan dengan kebutuhan skenario data, ukuran dataset, serta

keterbatasan sumber daya komputasi yang tersedia. Saran untuk penelitian selanjutnya disarankan menggunakan variasi ukuran data yag lebih besar, serta melakukan penukuran efisiensi memori dan waktu eksekusi secara langsung agar hasil analisis bersifat lebih komprehensif.

E. DAFTAR PUSTAKA

Al-aydi, B. M., & Abu-naser, S. S. (2025). *Comparative Study of Traditional and AI-Enhanced Sorting Algorithms : Comparative Study of Traditional and AI-Enhanced Sorting Algorithms : QuickSort , MergeSort , HeapSort , and TimSort*. (September). https://doi.org/10.13140/RG.2.2.13915.84005

Ali, H., Nawaz, H., Maitlo, A., & Soomro, I. (2021). Performance Analysis of Heap Sort and Insertion Sort Algorithm. *International Journal of Emerging Trends in Engineering Research*, *9*(5), 580–586. https://doi.org/10.30534/ijeter/2021/08952021

Ali, M. I., Fardiarsyah, R. D., Shodik, L., Kinanti, F. Z. D., & Pujiono, I. P. (2025). Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++. *LogicLink*, *2*(1), 1–17. https://doi.org/10.28918/logiclink.v2i1.10868

Amanda, S., Anastasya, L. D., Sulistia, F., Purnamasari, N., & Pujiono, I. P. (2026). Analisis Perbandingan Efisiensi Algoritma Introsort Dengan Algoritma Tradisional Bubble Sort dan Selection Sort. *JEIS: Jurnal Elektro Dan Informatika Swadharma*, *6*(1), 155–165. https://doi.org/10.56486/jeis.vol6no1.1086

Basir, R. R. (2020). Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java. *STRING (Satuan Tulisan Riset Dan Inovasi Teknologi)*, *5*(2), 109. https://doi.org/10.30998/string.v5i2.6250

Ghofur, A., Nazila, J., Holidiyah, I., Kholifah, S., Husaini Kulsum, F., Insiyah, N., Ibrahimy, U., Situbondo, K., & Timur, J. (2025). Penerapan Algoritma Insertion Sort Pada Aplikasi Pengolahan Data Mahasiswa Menggunakan Java Di Fakultas Sains Dan Teknologi Prodi Teknologi Informasi. *Eastasouth Journal of Positive Community Services*, *4*(01), 12–19. https://doi.org/10.58812/ejpcs.v4i01

Gudiato, C., Cahyaningtyas, C., & P., N. (2024). Decision Support System in Selecting Residential Houses using the S.A.W Method. and Fuzzy Logic. *G-Tech : Jurnal Teknologi Terapan*, *8*(1), 186–195. https://doi.org/10.33379/gtech.v8i1.3601

Hanafi, M. R., Faadhilah, M. A., Dwi Putra, M. T., & Pradeka, D. (2022). Comparison Analysis of Bubble Sort Algorithm with Tim Sort Algorithm Sorting Against the Amount of Data. *Journal of Computer Engineering, Electronics and Information Technology*, *1*(1), 29–38. https://doi.org/10.17509/coelite.v1i1.43794

Ilham, M. N., Setiawan, A. F., Kholifatun, I., Aldiansyah, M. H., & Pujiono, I. P. (2025). Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language. *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, *4*(3), 1950–1956. https://doi.org/10.59934/jaiea.v4i3.1051

Iskandar, I. D., Amirulloh, I., Pertiwi, M. W., Kusmira, M., Hikmah, A. B., & Supriadi,

D. (2020). Analysis of Bubble Sort and Insertion Sort Algorithm on Memory Efficiency Using Data Mining Approach. *Jurnal PILAR Nusa Mandiri*, *16*(1), 89–96. https://doi.org/10.33480/pilar.v16i1.1165

Isyqi, A., Astuti, G. D., Saputro, A. D., & Barryananda, M. A. (2024). Perbandingan Kinerja Algoritma Bubble Sort, Insertion Sort, Dan Selection Sort Menggunakan Data Bilangan Numerik Pada Program Python. *Seminar Nasional Sains Dan Teknologi "SainTek" Seri II*, *1*(2), 3047–6569. https://conference.ut.ac.id/index.php/saintek/article/view/2617

Latifah, R., Arriyanti, E., & Hakim, A. R. (2021). *Perbandingan Efisiensi Kinerja Algoritma Bubble Sort dan Algoritma Selection Sort Pada Parallel Programming* [STMIK Widya Cipta Dharma]. https://repository.wicida.ac.id/3687/

Mahrozi, N., & Faisal, M. (2023). Analisis Perbandingan Kecepatan Algoritma Selection Sort dan Bubble Sort. *Scientica: Jurnal Ilmiah Sains Dan Teknologi*, *1*(2), 89–98. https://doi.org/10.572349/scientica.v1i2.209

Musyaffa, M. Z., Raharjo, K., Faiz, M., Ammarulloh, S., & Pujiono, I. P. (2025). Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algoritma Pengurutan Tradisional Menggunakan Python. *JEIS: Jurnal Elektro Dan Informatika Swadharma*, *5*(2), 72–81. https://doi.org/10.56486/jeis.vol5no2.785

Pujiono, I. P., Kamal, M. R., Prayogi, A., Sari, C. A., & Ikhsanuddin, R. M. (2025). Algoritma Counting Sort Vs Algoritma Pengurutan Modern: Analisis Efisiensi Memori Dan Waktu Komputasi. *Jurnal Informatika Dan Teknik Elektro Terapan*, *13*(3). https://doi.org/10.23960/jitet.v13i3.6657

Sari, N., Gunawan, W. A., Sari, P. K., Zikri, I., & Syahputra, A. (2022). Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Dengan Menggunakan Bahasa Pemrograman Java. *ADI Bisnis Digital Interdisiplin Jurnal*, *3*(1), 16–23. https://doi.org/10.34306/abdi.v3i1.625

Sena, M. B., Hanun, R. M., Purnama, I. R., & Ardian, M. (2024). Perbandingan Kinerja Algoritma Sorting Dalam Pengurutan Data Menggunakan Bahasa Python. *Prosiding Seminar Nasional Sains Dan Teknologi Seri 02*, *1*(2), 310–314. https://conference.ut.ac.id/index.php/saintek/article/view/2633

Sutanto, D. S., Kirana, C., & Wahyuningsih, D. (2025). Analisis Efisiensi Waktu Bubble, Insertion, Merge, Dan Quick Sort Menggunakan Python. *Metik Jurnal*, *9*(1), 159–169. https://doi.org/10.47002/metik.v9i1.1053

Wijaya, S., Fauziah, & Harjanti, T. W. (2024). Perbandingan Algoritma Sorting dengan Menggunakan Bahasa Pemograman Javascript dalam Penggunaan Waktu Komputasi dan Penggunaan Memori. *STRING : Satuan Tulisan Riset Dan Inovasi Teknologi*, *8*(2), 294–302. https://doi.org/10.30998/string.v8i3.17972

Zulfa, M., Mikhael, M., & Sari, B. (2022). Analisis Perbandingan Algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam Mengurutkan Baris Angka Acak menggunakan Bahasa Java. *Jurnal Ilmiah Wahana Pendidikan*, *8*(13), 237–246. https://doi.org/10.5281/zenodo.6962346