

OPTIMASI PENGURUTAN DATA: PERBANDINGAN WAKTU EKSEKUSI *BUBBLE VS INSERTION VS HEAP SORT*

Naila Mifrotul Ula¹, Galang Satya Aulia², Rakhiel Ghina Suraya³, Alfiyan Firmansyah⁴, Imam Prayogo Pujiono⁵

Program Studi Bisnis Digital^{1,2,3,4}, Informatika⁵

Fakultas Ekonomi dan Bisnis Islam, Universitas Islam Negeri K.H. Abdurrahman Wahid Pekalongan

E-mail: * naila.mifrotul.ula25037@mhs.uingusdur.ac.id¹, galang.satya.aulia25013@mhs.uingusdur.ac.id², rakhiel.ghina.suraya25026@mhs.uingusdur.ac.id³, alfiyan.firmansyah25023@mhs.uingusdur.ac.id⁴, imam.prayogopujiono@uingusdur.ac.id⁵

Abstrak - Pengurutan data adalah proses penting dalam komputasi yang sangat berpengaruh terhadap kinerja sistem pengolahan informasi. Penelitian ini bertujuan untuk membandingkan efisiensi waktu eksekusi dari tiga algoritma pengurutan, yaitu Bubble Sort, Insertion Sort, dan Heap Sort. Metode yang digunakan adalah eksperimental dengan menerapkan ketiga algoritma tersebut dalam bahasa pemrograman C++. Pengujian dilakukan menggunakan dataset acak dengan ukuran 100, 1.000, dan 10.000 elemen. Waktu eksekusi dicatat menggunakan fungsi clock() agar hasilnya akurat dan dapat diukur. Hasil penelitian menunjukkan perbedaan performa yang signifikan. Bubble Sort memiliki performa terburuk, terutama untuk dataset besar dengan ukuran 10.000 elemen, di mana waktu eksekusinya melebihi 400 milidetik (ms). Insertion Sort menunjukkan performa yang lebih baik dibandingkan Bubble Sort terutama ketika menangani dataset kecil atau data yang sudah hampir terurut. Namun, algoritma yang terus memberikan performa terbaik adalah Heap Sort, karena mampu menjaga waktu eksekusi cepat dan stabil hanya sekitar 3-6 ms untuk mengurutkan 10.000 elemen. Pemilihan algoritma pengurutan yang tepat bergantung pada ukuran dan karakteristik data yang diproses. Penelitian ini juga memberikan data empiris yang bermanfaat bagi pengembang perangkat lunak dan akademisi dalam mengambil keputusan yang informatif tentang algoritma yang terbaik dan paling efisien untuk pengolahan data modern.

Kata Kunci: Algoritma; Bubble; Heap; Insertion Sort

I. PENDAHULUAN

Di era informasi saat ini, pengolahan data menjadi inti dari hampir semua sistem komputasi. Salah satu fungsi algoritma dalam dunia pemrograman yaitu algoritma pengurutan atau sorting algorithm, yang sering digunakan untuk proses suatu pengurutan data. Pengurutan data atau lebih dikenal dengan istilah *sorting* termasuk hal yang amat sangat penting bagi pengelolaan data. Karena dengan pengurutan proses pengelolaan data bias lebih cepat, lebih mudah, dan efisien (Astuti, 2023). Algoritma memasukkan elemen data ke kumpulan data lain dalam urutan tertentu. Proses pengurutan ini mengubah data yang sebelumnya disusun secara acak menjadi tersusun secara teratur sesuai dengan aturan tertentu (Saputro & Khasanah, 2018).

Algoritma pengurutan yang digunakan untuk penelitian ini yaitu Bubble Sort, Insertion Sort, dan Heap Sort. Algoritma Bubble Sort dan Insertion Sort mempunyai cara kerja yang sama yaitu dengan cara membandingkan elemen secara berurutan dan menukarnya hingga seluruh data berada dalam urutan yang benar. Sedangkan Heap Sort cara kerjanya menggunakan struktur data Heap (pohon biner lengkap) untuk mengurutkannya.

Meskipun penelitian tentang algoritma pengurutan sudah banyak dilakukan, tetap diperlukan analisis kinerja terhadap performa aktual ketiga algoritma ini menggunakan bahasa pemrograman C++ secara eksperimental. Berbeda dengan penelitian terdahulu yang cenderung hanya

meninjau kompleksitas waktu secara teoritis, penelitian ini menitikberatkan pada analisis empiris waktu eksekusi nyata dari algoritma Bubble Sort, Insertion Sort, dan Heap Sort dengan menggunakan implementasi langsung di bahasa pemrograman C++. Keaslian penelitian ini terletak pada pengukuran performa algoritma pada beragam ukuran data acak (100, 1.000, dan 10.000 elemen) serta lingkungan perangkat keras modern, sehingga hasil yang diperoleh lebih representatif terhadap kondisi pengolahan data aktual. Oleh karena itu, tujuan penelitian ini yaitu untuk menganalisis dan membandingkan efisiensi waktu antara Bubble Sort, Insertion Sort, dan Heap Sort serta memberikan gambaran mengenai algoritma yang lebih optimal digunakan pada berbagai ukuran data. Hasil penelitian ini diharapkan dapat digunakan sebagai referensi bagi pengembang sistem dan akademisi dalam memahami efisiensi algoritma pengurutan secara komprehensif.

II. TINJAUAN PUSTAKA

Algoritma Pengurutan

Algoritma pengurutan adalah cara mengatur data yang tadinya berantakan menjadi rapi dan teratur berdasarkan aturan tertentu (Sandria et al., 2022). Pengurutan memainkan peran penting dalam berbagai aplikasi seperti basis data, sistem pencarian informasi, dan algoritma lain yang membutuhkan data yang teratur dan terorganisir. Keakuratan dalam proses pengurutan tidak hanya

bergantung pada kompleksitas waktu, tetapi juga pada penggunaan memori, terutama ketika menangani data dalam jumlah besar (Zahwa et al., 2025).

Bubble Sort

Bubble Sort adalah metode algoritma pengurutan yang terus menukar dua elemen hingga proses pengurutannya selesai (Latifah et al., 2021). Bubble Sort melakukan perbandingan terus-menerus antara dua elemen yang berdekatan dalam Array. Jika kedua elemen tersebut tidak berada dalam urutan yang benar, maka posisi keduanya akan ditukar. Dalam setiap kali pengulangan, elemen terbesar akan berpindah ke posisi terakhir Array. Algoritma ini akan terus berjalan hingga semua elemen sudah terurut dengan benar, dan berhenti segera ketika menemukan bahwa data sudah dalam urutan yang sesuai (Rahmani, 2022). Misalnya, jika ada 7 elemen ($n = 7$), maka proses Bubble Sort akan melalui 6 tahapan (mulai dari 0 hingga $n - 2$) (Mahrozi & Faisal, 2023).

Insertion Sort

Insertion Sort adalah metode algoritma yang mengurutkan data dengan membandingkan setiap unsur dan memasukkan elemen tersebut ke tempat yang benar secara bertahap, sehingga cocok digunakan untuk tugas pengelolaan data yang tidak terlalu rumit (Ghofur et al., 2025). Metode Insertion bertujuan membuat bagian sisi kiri elemen diurutkan hingga seluruh elemen menjadi terurutkan (Saptadi & Sari, 2012). Untuk menemukan posisi yang benar, dilakukan pencarian secara berurutan dalam barisan elemen. Selama proses pencarian, elemen-elemen tersebut digeser secara perlahan hingga posisi yang benar tercapai (Tambunan et al., 2024). Contoh dari algoritma ini bisa kita temukan dalam kehidupan sehari-hari, misalnya saat mengurutkan kartu remi. Bayangkan ada sebuah meja yang berisi tumpukan kartu. Meja ini melambatkan kondisi elemen sebelum diurutkan. Berikut langkah-langkah pengurutannya:

1. Ambil kartu pertama dari meja, letakkan di tangan kiri.
2. Ambil kartu kedua dari meja, bandingkan dengan kartu di tangan kiri, lalu letakkan di posisi yang benar setelah membandingkan.
3. Ulangi langkah ini sampai semua kartu di meja berada dalam urutan yang tepat di tangan kiri.
4. Kartu di tangan kiri tersebut menunjukkan kondisi elemen setelah diurutkan. (Basir, 2020).

Heap Sort

Heap Sort merupakan salah satu algoritma pengurutan yang memiliki performa terbaik secara asimptotik, serta menggunakan pendekatan khusus dalam mengatasi masalah pengurutan, yaitu dengan memanfaatkan struktur pohon Heap (Chalidjen et

al., n.d.). Prosedur Heap Sort mengatur sekumpulan data yang disimpan dalam Array atau struktur pohon Heap. Cara kerjanya adalah dengan mengambil data dari node akar (posisi Array = 1) lalu menukar (*exchange*) dengan data node terakhir (posisi Array = indeks terbesar dalam pohon Heap). Setelah itu, node terakhir dihapus, dan Heap Sort memanggil prosedur Heapify untuk memastikan struktur pohon tetap memenuhi sifat Heap setelah pertukaran data tersebut. Hasil yang dikeluarkan adalah data yang telah terurut, bisa dari kecil ke besar (Ascending) atau dari besar ke kecil (Descending) (Pseudocode et al., 2015).

Teori Kompleksitas Waktu

Teori kompleksitas waktu merupakan bagian penting dalam analisis algoritma. Notasi O-Besar tidak selalu mewakili algoritma yang tercepat pada data tertentu, tetapi pendekatan ini masih sangat efektif untuk membandingkan berbagai algoritma ketika ukuran data masukan semakin besar (Subandijo, 2011). Dengan memahami kompleksitas waktu dari Bubble Sort, Insertion Sort, dan Heap Sort kita dapat membuat prediksi yang lebih baik tentang performa algoritma dalam situasi nyata. Seperti apa yang dinyatakan oleh (Winda Noviana, 2022), algoritma yang baik bisa membuat program berjalan lebih cepat, sehingga memilih algoritma yang tepat sangat penting, terutama untuk aplikasi yang menangani banyak jumlah data.

III. METODE PENELITIAN

Penelitian ini menggunakan metode kuantitatif dengan pendekatan eksperimental. Penelitian yang kami analisis berpusat pada cara pengumpulan data dengan menggunakan sebuah program. Di dalam program tersebut, terdapat sebuah Array, kemudian nilai-nilai yang akan diatur adalah angka-angka acak yang disimpan di dalam Array. Array merupakan kumpulan data yang digunakan untuk menyimpan beberapa item dengan tipe data yang sama, biasanya digunakan untuk mengatur urutan data (Bagas & Wantoro, 2025). Tujuan dari penelitian ini untuk mengetahui hasil kerja perbandingan dari ke tiga algoritma pengurutan, yaitu Bubble Sort, Insertion Sort, dan Heap Sort terutama dalam hal waktu yang diperlukan dalam menyelesaikan pengurutan, khususnya berdasarkan jumlah perbandingan yang dilakukan saat mengurutkan sejumlah data tertentu. Rentang ini dipilih agar mempermudah distribusi data sekaligus menguji kemampuan algoritma dalam kondisi nilai yang terbatas dan mungkin mengandung banyak salinan yang sama, sehingga evaluasi mengenai efisiensi waktu bisa dilakukan secara lebih konsisten (Irfan Ali et al., 2025).

Dalam penelitian ini, bahasa pemrograman yang digunakan adalah C++, yang dikenal karena efisiensinya dan kemampuannya untuk menangani

berbagai jenis data. Spesifikasi laptop yang digunakan dalam pengujian ini adalah:

1. Windows 11 64-Bit
2. Memory: 8 GB RAM
3. Processor: Intel Core i3-1115G4
4. SSD berkapasitas 512 GB
5. Compiler: Code::Blocks

Penggunaan pemrograman C++ memungkinkan penulis untuk mengimplementasikan algoritma dengan efisien dan mengukur waktu eksekusi dengan akurat. Desain eksperimen ini melibatkan pembuatan tiga program terpisah untuk mengimplementasikan algoritma Bubble Sort, Insertion Sort, dan Heap Sort. Dataset yang digunakan terdiri dari elemen-elemen acak dengan ukuran 100, 1.000, dan 10.000. Pengujian dilakukan dengan menjalankan masing-masing algoritma pada dataset yang sama dan mencatat waktu eksekusi untuk setiap ukuran data.

Pada penelitian ini dilakukan pengujian efisiensi waktu algoritma Bubble Sort menggunakan bahasa C++. Data yang digunakan berupa bilangan acak yang dihasilkan dengan fungsi `rand()` dari pustaka `<cstdlib>` dan disimpan dalam *Array* dinamis agar dapat menyesuaikan jumlah elemen. Waktu eksekusi diukur menggunakan fungsi `clock()` dari pustaka `<ctime>`, yang menghitung lama waktu CPU selama proses pengurutan berlangsung. Selisih antara waktu mulai dan selesai kemudian dikonversi ke milidetik (ms) dengan membagi hasilnya terhadap konstanta `CLOCKS_PER_SEC`. Coding untuk mengukur waktu eksekusi menggunakan Bubble Sort adalah sebagai berikut:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        bool swapped = false;
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
            if (!swapped)
                break;
        }
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    // Ubah nilai n di bawah untuk percobaan: 100, 1000, atau 10000
    int n = 10000;
    int *arr = new int[n];
    srand(time(0));
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 10000 + 1;
    cout << "Jumlah elemen: " << n << endl;
    clock_t start = clock();
    bubbleSort(arr, n);
    clock_t end = clock();
    double time_taken_ms = double(end - start) / CLOCKS_PER_SEC * 1000;
    cout << "Waktu eksekusi Bubble Sort: " << time_taken_ms << " milidetik (ms)" << endl;
    delete[] arr;
    return 0;
}
```

Gambar 1. Codingan Bubble Sort

Penelitian ini mengukur efisiensi waktu eksekusi dari algoritma Insertion Sort dalam bahasa pemrograman C++. Setiap elemen data berupa bilangan bulat yang dihasilkan secara acak menggunakan fungsi `rand()` dari pustaka `<cstdlib>`. Proses pengukuran waktu eksekusi dilakukan menggunakan fungsi `clock()` dari pustaka `<ctime>` yang berfungsi untuk

menghitung waktu CPU yang digunakan oleh program dalam satuan clock ticks. Dengan membagi jumlah clock ticks yang diperoleh dengan konstanta `CLOCKS_PER_SEC`, waktu eksekusi dapat dikonversi menjadi satuan detik. Agar hasil lebih mudah dipahami, waktu tersebut kemudian dikalikan dengan 1000 untuk mendapatkan nilai dalam milidetik (ms). Coding untuk mengukur waktu eksekusi menggunakan Insertion Sort adalah sebagai berikut:

```
#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

void insertionSort(int arr[], int n) {
    for (int i = 1; i < n; i++) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j--;
        }
        arr[j + 1] = key;
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    // Ubah nilai n di bawah untuk percobaan: 100, 1000, atau 10000
    int n = 10000;
    int *arr = new int[n];
    srand(time(0));
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 10000 + 1;
    cout << "Jumlah elemen: " << n << endl;
    clock_t start = clock();
    insertionSort(arr, n);
    clock_t end = clock();
    double time_taken_ms = double(end - start) / CLOCKS_PER_SEC * 1000;
    cout << "Waktu eksekusi Insertion Sort: " << time_taken_ms << " milidetik (ms)" << endl;
    delete[] arr;
    return 0;
}
```

Gambar 2. Codingan Insertion Sort

Dalam penelitian ini, efisiensi waktu algoritma Heap Sort diuji menggunakan pemrograman C++. Data yang digunakan adalah bilangan acak yang disimpan dalam *Array* dinamis dengan jumlah elemen *n* (misalnya, 10.000 elemen). Pengukuran waktu eksekusi dilakukan dengan presisi menggunakan fungsi `clock()` dari pustaka `<ctime>`. Waktu mulai dicatat sebelum memanggil fungsi `72eapso()`, sedangkan waktu akhir dicatat setelah proses pengurutan selesai. Selisih waktu CPU yang terukur (`end - start`) kemudian diubah menjadi milidetik (ms), memberikan ukuran kuantitatif langsung mengenai performa algoritma tersebut. Secara teori, algoritma ini memiliki tingkat kecepatan eksekusi sebesar $O(n \log n)$. Coding untuk mengukur waktu eksekusi menggunakan Heap Sort adalah sebagai berikut:

```

#include <iostream>
#include <ctime>
#include <cstdlib>
using namespace std;

void heapify(int arr[], int n, int i) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;
    if (left < n && arr[left] > arr[largest])
        largest = left;
    if (right < n && arr[right] > arr[largest])
        largest = right;
    if (largest != i) {
        int temp = arr[i];
        arr[i] = arr[largest];
        arr[largest] = temp;
        heapify(arr, n, largest);
    }
}

void heapSort(int arr[], int n) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i = n - 1; i > 0; i--) {
        int temp = arr[0];
        arr[0] = arr[i];
        arr[i] = temp;
        heapify(arr, i, 0);
    }
}

void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

int main() {
    // ubah nilai n di bawah untuk percobaan : 100, 1000, 10000
    int n = 10000;
    int *arr = new int[n];
    srand(time(0));
    for (int i = 0; i < n; i++)
        arr[i] = rand() % 10000 + 1;
    cout << "Jumlah elemen: " << n << endl;
    clock_t start = clock();
    heapSort(arr, n);
    clock_t end = clock();
    double time_taken_ms = double(end - start) / CLOCKS_PER_SEC * 1000;
    cout << "Waktu eksekusi Heap Sort: " << time_taken_ms << " milidetik (ms)" << endl;
    delete[] arr;
    return 0;
}

```

Gambar 3. Codingan Heap Sort

IV. HASIL DAN PEMBAHASAN

Insertion Sort dan Bubble Sort dianggap kurang cepat karena keduanya membutuhkan waktu yang berbanding lurus dengan kuadrat jumlah data, yaitu $O(n^2)$. Hal ini menyebabkan kecepatan algoritma menurun secara signifikan ketika jumlah data semakin banyak. Di sisi lain, Heap Sort memiliki kecepatan yang lebih baik karena kompleksitas waktu pada semua kondisi adalah $O(n \log n)$, tetapi algoritma ini juga memiliki kekurangan, seperti proses pembuatan Heap yang memerlukan waktu tambahan dan membuatnya tidak selalu lebih cepat dibandingkan algoritma lain dalam penggunaan nyata. Hasil pengujian waktu eksekusi masing-masing algoritma akan ditampilkan melalui ilustrasi dan table agar proses analisis menjadi lebih mudah.

Pada pengujian dengan dataset yang terdiri dari 100 elemen, ketiga algoritma yaitu *Bubble Sort*, *Insertion Sort*, dan *Heap Sort* menunjukkan hasil yang sangat cepat, sehingga waktu yang dibutuhkan untuk menjalankannya tercatat 0 ms. Situasi ini menunjukkan bahwa untuk ukuran dataset seperti ini, perbedaan efisiensi waktu antara ketiga algoritma belum terlihat secara jelas maupun bisa diukur melalui penghitungan waktu eksekusi.

```

Jumlah elemen: 100
Waktu eksekusi Bubble Sort: 0 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.

```

Gambar 4. Hasil uji dengan dataset berukuran 100 elemen untuk Algoritma Bubble Sort

```

Jumlah elemen: 100
Waktu eksekusi Insertion Sort: 0 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.028 s
Press any key to continue.

```

Gambar 5. Hasil uji dengan dataset berukuran 100 elemen untuk Algoritma Insertion Sort

```

Jumlah elemen: 100
Waktu eksekusi Heap Sort: 0 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.

```

Gambar 6. Hasil uji dengan dataset berukuran 100 elemen untuk Algoritma Heap Sort

Ketika ukuran dataset meningkat menjadi 1.000 elemen, perbedaan waktu eksekusi antaralgoritma mulai terlihat secara lebih signifikan. Bubble Sort menunjukkan waktu eksekusi yang lebih tinggi, dengan selisih 2 milidetik dibandingkan Insertion Sort dan 4 milidetik lebih lambat dibandingkan Heap Sort. Sementara itu, Heap Sort mampu mempertahankan kinerjanya dengan waktu eksekusi tetap sebesar 0 milidetik meskipun jumlah elemen meningkat hingga 1.000.

```

Jumlah elemen: 1000
Waktu eksekusi Bubble Sort: 4 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.

```

Gambar 7. Hasil uji dengan dataset berukuran 1000 elemen untuk Algoritma Bubble Sort

```

Jumlah elemen: 1000
Waktu eksekusi Insertion Sort: 2 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.032 s
Press any key to continue.

```

Gambar 8. Hasil uji dengan dataset berukuran 1000 elemen untuk Algoritma Insertion Sort

```

Jumlah elemen: 1000
Waktu eksekusi Heap Sort: 0 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.025 s
Press any key to continue.

```

Gambar 9. Hasil uji dengan dataset berukuran 1000 elemen untuk Algoritma Heap Sort

Dalam pengujian menggunakan dataset yang terdiri dari 10.000 elemen, hasil menunjukkan bahwa Bubble Sort membutuhkan waktu eksekusi yang sangat lama, yaitu 413 ms. Sementara itu, Insertion Sort membutuhkan waktu yang lebih singkat, meskipun masih dianggap tidak terlalu cepat, yaitu 196 ms. Di sisi lain, Heap Sort menunjukkan performa terbaik dengan waktu eksekusi yang sangat cepat, hanya 5 ms saja.

```
Jumlah elemen: 10000
Waktu eksekusi Bubble Sort: 413 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.445 s
Press any key to continue.
```

Gambar 10. Hasil uji dengan dataset berukuran 10.000 elemen untuk Algoritma Bubble Sort

```
Jumlah elemen: 10000
Waktu eksekusi Insertion Sort: 196 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.223 s
Press any key to continue.
```

Gambar 11. Hasil uji dengan dataset berukuran 10.000 elemen untuk Algoritma Insertion Sort

```
Jumlah elemen: 10000
Waktu eksekusi Heap Sort: 5 milidetik (ms)

Process returned 0 (0x0)   execution time : 0.035 s
Press any key to continue.
```

Gambar 12. Hasil uji dengan dataset berukuran 10.000 elemen untuk Algoritma Heap Sort

Dataset elemen lengkap hasil pengujian masing-masing algoritma bisa diakses melalui link berikut ini :

<https://bit.ly/bubblesort100elemen>
<https://bit.ly/bubblesort1000elemen>
<https://bit.ly/bubblesort10000elemen>
<https://bit.ly/insertionsort100elemen>
<https://bit.ly/insertionsort1000elemen>
<https://bit.ly/insertionsort10000elemen>
<https://bit.ly/Heapsort100elemen>
<https://bit.ly/Heapsort1000elemen>
<https://bit.ly/Heapsort10000elemen>

Penelitian ini menunjukkan bahwa Bubble Sort dan Insertion Sort lebih sesuai diterapkan pada dataset berukuran kecil hingga menengah. Hal ini sejalan dengan hasil penelitian (Barokah et al., 2025) yang mengungkapkan bahwasannya Insertion Sort secara konsisten menunjukkan kinerja lebih unggul daripada Bubble Sort, baik dari aspek waktu eksekusi maupun efisiensi jumlah perbandingan, terutama ketika data yang diuji berukuran kecil hingga menengah. Selain itu, Insertion Sort juga lebih efektif digunakan pada data yang telah terurut sebagian. Berdasarkan hasil tersebut, algoritma Insertion Sort direkomendasikan untuk diterapkan pada sistem informasi berskala kecil dan menengah. Sedangkan Heap Sort cocok untuk dataset berskala besar. Hal ini sejalan dengan hasil penelitian (Pujiono et al., 2025) yang mengungkapkan bahwa metode Heap Sort ini efisien dengan waktu kerja $O(n \log n)$, sehingga tepat digunakan untuk mengurutkan data dalam jumlah yang besar.

Tabel 1. Hasil uji dalam algoritma Bubble Sort

Bubble Sort		
Percobaan Ke	Ukuran Data	Durasi Eksekusi (milidetik)
1	100	0
	1000	4
	10000	413
2	100	0
	1000	5
	10000	400
3	100	0
	1000	7
	10000	351

Tabel 2. Hasil uji algoritma Insertion Sort

Insertion Sort		
Percobaan ke	Ukuran Data	Durasi Eksekusi (milidetik)
1	100	0
	1000	2
	10000	196
2	100	0
	1000	2
	10000	162
3	100	0
	1000	6
	10000	207

Tabel 3. Hasil uji algoritma Heap Sort

Heap Sort		
Percobaan Ke	Ukuran Data	Durasi Eksekusi (milidetik)
1	100	0
	1000	0
	10000	5
2	100	0
	1000	1
	10000	3
3	100	0
	1000	1
	10000	6

Data dalam tabel ini disusun berdasarkan hasil eksperimen yang dilakukan dalam penelitian ini.

Berdasarkan ketiga tabel diatas, dapat disimpulkan bahwa Insertion Sort masih memiliki sedikit keunggulan dibandingkan Bubble Sort, terutama ketika data hampir terurut. Data dalam tabel menunjukkan bahwa Insertion Sort cenderung menghasilkan waktu eksekusi yang lebih stabil dan lebih rendah dibandingkan Bubble Sort. Ini

disebabkan oleh mekanisme penyisipan yang lebih efisien dan jumlah pertukaran yang lebih sedikit, sehingga Insertion Sort lebih cocok dengan perubahan pola data.

Bubble Sort merupakan algoritma yang paling tidak efisien, terutama saat mengolah data dalam jumlah besar. Banyaknya proses pertukaran elemen yang terus-menerus menyebabkan waktu eksekusinya jauh lebih lama, seperti terlihat dari durasi pengurutan data sebanyak 10.000 elemen yang mencapai lebih dari 300-400 ms lebih.

Di sisi lain, Heap Sort menunjukkan performa terbaik dari ketiga algoritma tersebut. Dengan kompleksitas $O(n \log n)$, Heap Sort mampu memproses data baik yang kecil maupun besar dengan waktu eksekusi yang sangat stabil dan rendah. Pada data 10.000 elemen, Heap Sort hanya membutuhkan waktu 3-6 ms, jauh lebih cepat dibandingkan dua algoritma lainnya. Efisiensi ini berasal dari struktur Heap yang secara sistematis mengatur elemen-elemen sehingga proses pengurutan berjalan optimal.

Dengan hasil temuan ini, penelitian menggambarkan dengan jelas mengenai performa ke tiga algoritma dalam konteks pengurutan data. Hasil ini dapat menjadi acuan bagi pengembang perangkat lunak dalam memilih algoritma yang tepat berdasarkan kebutuhan aplikasi dan jenis data yang akan diolah.

V. KESIMPULAN DAN SARAN

Kesimpulan

Penelitian ini menunjukkan bahwa ketiga algoritma pengurutan Bubble Sort, Insertion Sort, dan Heap Sort menunjukkan perbedaan yang cukup besar, terutama ketika digunakan untuk data berukuran besar. Bubble Sort dipandang sebagai algoritma yang paling tidak efisien, karena waktu yang dibutuhkan untuk menyelesaikan tugas meningkat secara drastis seiring bertambahnya jumlah data. Insertion Sort memiliki performa lebih unggul daripada Bubble Sort, terutama pada data yang ukurannya kecil hingga sedang atau data yang sudah terurut sebagian. Cara kerjanya dengan menyisipkan elemen ke posisi yang tepat membuat jumlah pertukaran lebih sedikit, sehingga waktu eksekusinya lebih stabil.

Heap Sort menunjukkan performa terbaik berdasarkan semua uji coba yang dilakukan. Dengan kompleksitas $O(n \log n)$, algoritma ini mampu menyelesaikan pengurutan dalam waktu yang paling cepat dan tetap stabil, bahkan pada data sebanyak 10.000 elemen.

Dari hasil eksperimen tersebut, pemilihan algoritma pengurutan harus mempertimbangkan besar dan karakteristik data. Untuk data yang besar, disarankan menggunakan Heap Sort, sedangkan untuk data kecil hingga sedang, Insertion Sort bisa menjadi pilihan yang efisien. Bubble Sort tidak

disarankan digunakan untuk pengurutan skala besar karena waktu eksekusinya terlalu lama.

Saran

Rekomendasi untuk penelitian selanjutnya adalah untuk mengeksplorasi algoritma pengurutan lain, seperti Quick Sort, Merge Sort, atau Radix Sort yang dikenal lebih efisien dalam banyak situasi. Penelitian lebih lanjut juga dapat dilakukan dengan membandingkan algoritma-algoritma tersebut dalam konteks yang lebih luas, termasuk faktor-faktor lain seperti penggunaan memori dan kompleksitas ruang. Hal ini akan memberikan pengetahuan yang lebih mendalam bagi para peneliti dan praktisi dalam memilih algoritma pengurutan yang lebih tepat untuk berbagai aplikasi.

DAFTAR PUSTAKA

- Astuti, Y. A. (2023). Analisis Pengujian Data Algoritma Bubble Sort. *REMIK: Riset Dan E-Jurnal Manajemen Informatika Komputer*, 7(3), 1413-1420. <https://polgan.ac.id/jurnal/index.php/remik/article/view/12594>
- Bagas, A., & Wantoro, F. (2025). *Analisis Perbandingan Efisiensi Algoritma Bubble Sort Dan. January*.
- Barokah, M. R. S., Saputra, T., Sutabri, T., & Darma, B. (2025). Perbandingan Kinerja Algoritma Bubble Sort dan Insertion Sort dalam Pengurutan Data Penjualan UMKM. *MIFORTEKH (Jurnal Manajemen Informatika & Teknologi)*, 5(1), 184-195. <https://journal.stiestekom.ac.id/index.php/mifortekh>
- 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>
- Chalikdjen, E., Ong, H., & Sajuthi, S. P. (n.d.). Heap Tree dan Kegunaannya dalam Heap Sort. *Https://Informatika.Stei.Itb.Ac.Id*, 1-5. <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/Makalah/MakalahStmik08.pdf>
- Ghofur, A., Nazila, J., Holidiyah, imatul, 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>
- Irfan Ali, M., Rangga Dzikri Fardiarsyah, Lukman Shodik, Fadilah Zahra Dwi Kinanti, & Imam

- Prayogo Pujiono. (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>
- Latifah, R., Arriyanti, E., & Hakim, A. R. (2021). *Perbandingan Efisiensi Kinerja Algoritma Bubble Sort dan Algoritma Selection Sort Pada Parallel Programming*. 1–78.
- Mahrozi, N., & Faisal, M. (2023). Analisis Perbandingan Kecepatan Algoritma Selection Sort Dan Bubble Sort. *Jurnal Ilmiah Sain Dan Teknologi*, 1(2), 89–98.
- Pseudocode, J., Wijaya, A., & Feter, N. (2015). *APLIKASI SIMULASI PENGURUTAN DATA. II*(September), 81–88.
- Pujiono, I. P., Rachmawanto, E. H., & Winarsih, N. A. S. (2025). Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis. *Jurnal Manajemen Informatika (JAMIKA)*, 15(1), 47–59. <https://doi.org/10.34010/jamika.v15i1.13230>
- Rahmani, M. K. I. (2022). Smart Bubble Sort: A Novel and Dynamic Variant of Bubble Sort Algorithm. *Computers, Materials and Continua*, 71(2), 4895–4913. <https://doi.org/10.32604/cmc.2022.023837>
- Sandria, Y. A., Nurhayoto, M. R. A., Ramadhani, L., Harefa, R. S., & Syahputra, A. (2022). Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP. *Hello World Jurnal Ilmu Komputer*, 1(4), 190–194. <https://doi.org/10.56211/helloworld.v1i4.187>
- Saptadi, A. H., & Sari, D. W. (2012). Analisis Algoritma Insertion Sort, Merge Sort Dan Implementasinya Dalam Bahasa Pemrograman C++. *JURNAL INFOTEL - Informatika Telekomunikasi Elektronika*, 4(2), 10. <https://doi.org/10.20895/infotel.v4i2.103>
- Saputro, F. E., & Khasanah, F. N. (2018). Teknik Selection Sort dan Bubble Sort Menggunakan Borland C++. *Jurnal Mahasiswa Bina Insani*, 2(2), 136–145. <http://ejournal-binainsani.ac.id/index.php/JMBI/article/view/916>
- Subandijo. (2011). Efisiensi Algoritma dan Notasi O-Besar Landasan Teori. *Computer Science Department, School of Computer Science Binus University*, 2(9), 849–858.
- Tambunan, V. F., Akva, S., Davina, S., Indra, Z., & Medan, U. N. (2024). *Analisis Kinerja Insertion Sort Dan Selection Sort Dalam Pemrograman Python*. 8(12), 33–38.
- Winda Noviana, R. A. (2022). Analisis Dan Optimalisasi Algoritma Kinerja Algoritma Pemrograman Dan Struktur Data Dalam Penyelesaian Masalah Komputasi. *Adil: STIH YPM Law Journal*, 4(2), 158–164.
- Zahwa, S., Nailah Dhina Amelia, Rizqi Nafila, Rosila Agustina Putri, & Imam Prayogo Pujiono. (2025). Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++. *Jurnal RESTIKOM: Riset Teknik Informatika Dan Komputer*, 7(1), 123–136. <https://doi.org/10.52005/restikom.v7i1.428>