# ANALISIS EMPIRIS PENGARUH UKURAN DATA TERHADAP WAKTU DAN MEMORI PADA ALGORITMA SORTING MENGGUNAKAN C++

**Aura Air Virdaos[1], Aulia Sari Ningsih[2], Fatikhatul Kamilah[3], Zuhailiyatus Syifa Eda[4], Imam Prayogo Pujiono[5]**
aura.air.virdaos25027@mhs.uingusdur.ac.id[1], aulia.sari.ningsih25020@mhs.uingusdur.ac.id[2], fatikhatul.kamilah25018@mhs.uingusdur.ac.id[3], zuhailiyatus.syifa.eda25039@mhs.uingusdur.ac.id[4], imam.prayogopujiono@uingusdur.ac.id[5]
**UIN K.H. Abdurrahman Wahid Pekalongan**

## ABSTRAK

Penelitian ini bertujuan untuk menganalisis dan membandingkan waktu eksekusi serta penggunaan memori tiga algoritma pengurutan data, yaitu Bubble Sort, Selection Sort, dan Insertion Sort, yang diimplementasikan menggunakan bahasa pemrograman C++. Setiap algoritma diuji dengan tiga variasi ukuran data, yaitu 100, 1.000, dan 10.000 elemen, untuk melihat bagaimana peningkatan jumlah data memengaruhi performa algoritma. Dua metrik utama yang diukur adalah waktu eksekusi (dalam nanodetik) dan penggunaan memori (dalam byte) selama proses pengurutan berlangsung. Hasil eksperimen menunjukkan bahwa semakin besar ukuran data, semakin lama waktu pemrosesan yang dibutuhkan oleh ketiga algoritma, sejalan dengan kompleksitas waktu $O(n^2)$ yang dimilikinya. Dari sisi performa, Selection Sort menghasilkan waktu eksekusi yang relatif cepat dengan penggunaan memori yang stabil, sehingga menjadi algoritma paling seimbang pada lingkungan pengujian ini. Bubble Sort berada pada posisi menengah dengan kinerja yang masih cukup baik, sedangkan Insertion Sort membutuhkan waktu paling lama ketika ukuran data meningkat, meskipun tetap efisien untuk data berukuran kecil atau data yang sudah hampir terurut. Temuan ini memberikan gambaran empiris mengenai trade-off antara kecepatan dan penggunaan memori pada algoritma pengurutan klasik dan dapat dijadikan acuan dalam pemilihan metode pengurutan untuk skenario pengolahan data sederhana.
**Kata Kunci**: Algoritma Pengurutan, Efisiensi Waktu, Penggunaan Memori, C++, Data.

## *ABSTRACT*

*The research is aimed at analyzing and comparing execution times and memory usage of three data-sorting algorithms, which are bubble sort, selection sort, and graphic sort, implemented using C ++ programming language. Each algorithm is tested with three variations in data size, which is 100, 1.000 and 10.000 elements, to see how increasing amounts of data affect the algorithm's performance. Two main metrics measured are execution times (in nanoseconds) and memory use (in bytes) during the sequencing process. Experiments have shown that the larger the size of data, the longer the processing time required by all three algorithms, aligns with the complexity of his o (n) time. From the performance side, selection sort is producing relatively fast execution times with stable memory usage, providing the most balanced algorithms in the testing environment. Bubble sort is ata medium position with a reasonably good performance, whereas attempting esprs take the longest when data size increases, although it remains efficient for small data or already virtually capsized data. This finding gives an empirical picture of a trade-off between the speed and memory usage on a classical sorting algorithm and can be referenced in selecting a sorting method for a simple data processing scenario.*
***Keywords**: Sequencing Algorithms, Time Efficiency, Memory Use, C ++.*

## PENDAHULUAN

Kemajuan teknologi informasi telah mengubah cara data dikumpulkan, disimpan, dan diolah, dari metode tradisional menjadi sistem digital berbasis komputasi awan dan infrastruktur komputasi berskala besar (Pujiono et al., 2024; Ali et al., 2025). Pertumbuhan

volume data yang masif menuntut teknik pengurutan (sorting) yang efisien agar proses pencarian, analisis, dan pengambilan keputusan dapat dilakukan secara cepat dan akurat (Knuth, 1998; Syafnidawaty, 2020; Yagci & Mishra, 2016). Dalam konteks ini, pemilihan algoritma pengurutan yang tepat menjadi aspek penting dalam perancangan perangkat lunak, sistem basis data, maupun sistem informasi berskala besar.

Sorting atau pengurutan merupakan proses mengatur elemen-elemen dari suatu himpunan data acak sehingga menjadi terurut berdasarkan kriteria tertentu, misalnya dari nilai terkecil ke terbesar (Anonim, 2011). Terdapat banyak algoritma pengurutan yang telah dikembangkan, seperti Heap Sort, Quick Sort, Shell Sort, Merge Sort, Selection Sort, Insertion Sort, dan Bubble Sort (Knuth, 1998; Munir & Lidya, 2016; Ullah, 2016). Meskipun ketiganya (Bubble Sort, Selection Sort, dan Insertion Sort) memiliki kompleksitas waktu $O(n^2)$, karakteristik implementasi dan pola akses memorinya dapat menghasilkan performa yang berbeda pada lingkungan dan ukuran data tertentu.

Berbagai penelitian telah menganalisis dan membandingkan performa algoritma pengurutan. Arifin dan Setiyadi (2020), misalnya, membandingkan Bubble Sort dengan Quick Sort menggunakan C++ dan menunjukkan bahwa Quick Sort memiliki waktu eksekusi yang lebih baik. Rahayuningsih (2016) melaporkan bahwa Bubble Sort membutuhkan waktu sekitar 6,84 detik untuk mengurutkan 250 data pada aplikasi Visual Basic 6.0, sedangkan algoritma lain hanya memerlukan 0,11 detik. Saputro dan Khasanah (2018) menguji efisiensi Bubble Sort dengan C++ dan menemukan bahwa algoritma tersebut memerlukan waktu sekitar 0,094 detik untuk mengurutkan 100 data. Penelitian lain membahas perbandingan beberapa algoritma sekaligus, misalnya Insertion Sort, Merge Sort, dan Quick Sort (Pratama et al., 2018), maupun analisis efisiensi memori dan waktu pada banyak algoritma pengurutan dalam berbagai bahasa pemrograman (Pujiono et al., 2024; Ali et al., 2025).

Meskipun telah banyak studi komparatif, sebagian besar penelitian berfokus pada satu metrik saja (misalnya waktu eksekusi) atau membandingkan banyak algoritma sekaligus tanpa analisis mendalam terhadap perilaku setiap algoritma $O(n^2)$ pada variasi ukuran data yang terkontrol. Selain itu, studi yang secara eksplisit mengukur dan mendiskusikan penggunaan memori proses (working set size) secara empiris pada lingkungan C++ di Windows masih relatif terbatas.

Berdasarkan pemetaan tersebut, penelitian ini menawarkan tiga kontribusi utama. Pertama, penelitian ini menyajikan analisis empiris terkontrol terhadap tiga algoritma pengurutan klasik berkompleksitas $O(n^2)$ (Bubble Sort, Selection Sort, dan Insertion Sort) dengan dua metrik sekaligus, yaitu waktu eksekusi pada tingkat nanodetik dan penggunaan memori proses (working set size) dalam satuan byte pada lingkungan C++ di Windows 11. Kedua, penelitian ini membandingkan pola skala kinerja pada tiga ukuran data yang representatif (kecil, menengah, besar), sehingga memberikan gambaran praktis batas keefektifan setiap algoritma. Ketiga, hasil eksperimen dipetakan terhadap teori kompleksitas dan temuan penelitian sebelumnya, disertai penyediaan kode sumber dan dataset uji secara terbuka, sehingga dapat dimanfaatkan sebagai bahan ajar maupun dasar penelitian lanjutan dalam analisis algoritma pengurutan.

## METODOLOGI

Penelitian menggunakan metode eksperimen komputasional, yakni metode yang dilakukan dengan menjalankan pengujian langsung melalui program komputer untuk mengamati kinerja beberapa algoritma pengurutan data. Peneliti menggunakan cara penelitian yang dimulai dengan membuat dataset berisi angka-angka acak dari rentang nilai 1 sampai 99 untuk diuji. Dataset ini dibagi menjadi tiga kelompok berdasarkan jumlah data

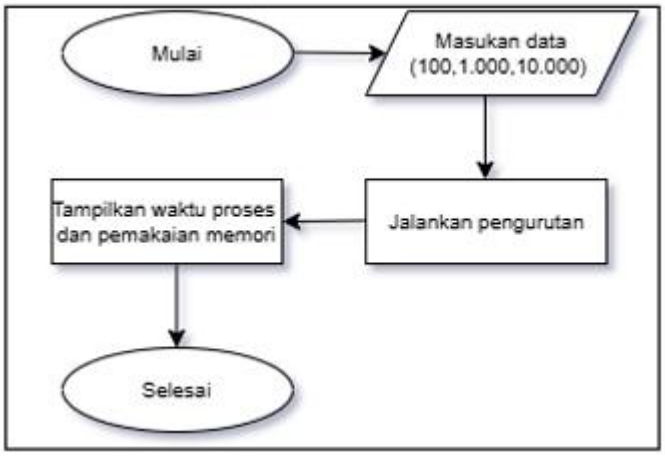
yang ada: 100 data untuk dataset skala kecil, 1.000 data untuk dataset skala menengah, dan 10.000 data untuk dataset skala besar, sesuai dengan sintaksis program yang digunakan untuk membuat dataset tersebut. Contoh kode yang digunakan adalah sebagai berikut:

```
// Fungsi untuk mendapatkan penggunaan memori (dalam byte)
SIZE_T getMemoryUsage() {
PROCESS_MEMORY_COUNTERS_EX pmc;
GetProcessMemoryInfo(GetCurrentProcess(),
(PROCESS_MEMORY_COUNTERS*)&pmc, sizeof(pmc));
return pmc.WorkingSetSize;
}
int main() {
const int n = 1000; // jumlah elemen array
vector<int> data(n);
srand(time(0));
// Isi array dengan nilai acak
for (int i = 0; i < n; i++) {
data[i] = rand() % 100000;
}
// Pengukuran memori sebelum proses
SIZE_T memBefore = getMemoryUsage();

// Pengukuran waktu mulai
auto start = high_resolution_clock::now();
// Pengukuran waktu selesai
auto stop = high_resolution_clock::now();
// Pengukuran memori setelah proses
SIZE_T memAfter = getMemoryUsage();
// Hitung waktu eksekusi (dalam nanosekon)
auto duration = duration_cast<nanoseconds>(stop - start).count();
// Hitung perbedaan memori
SIZE_T memUsed = (memAfter > memBefore) ? (memAfter - memBefore) : 0;
```

Setelah dataset berhasil dibuat,Sebelum proses pengurutan dimulai, program memanggil fungsi getMemoryUsage() yang menggunakan API Windows GetProcessMemoryInfo untuk mencatat jumlah memori yang digunakan oleh proses program dalam satuan byte. langkah berikutnya adalah menerapkan beberapa algoritma pengurutan ke dataset tersebut. Algoritma-algoritma ini berjalan dengan mengatur data dari angka terkecil sampai angka terbesar, sehingga memungkinkan kita membandingkan seberapa efisien mereka dalam mengelola berbagai macam jumlah data. Setelah sorting selesai, program kembali mengukur penggunaan memori (memAfter) dan menghitung perbedaannya terhadap nilai awal (memBefore) guna mengetahui tambahan memori yang digunakan selama proses berlangsung. Untuk mengetahui kinerja algoritma, penelitian ini mencatat waktu dan penggunaan memori sebelum, selama, serta setelah proses pengurutan dilakukan. Waktu eksekusi dihitung dengan mengukur selisih antara waktu sebelum dan sesudah algoritma dijalankan menggunakan fungsi std::chrono::high_resolution_clock::now() dari pustaka chrono, yang bisa memberikan akurasi hingga nanodetik dan memberikan gambaran jelas mengenai seberapa cepat proses pengurutan berlangsung. Penggunaan memori diukur secara kasar berdasarkan ukuran kontainer data yang dialokasikan menggunakan fungsi sizeof() serta pemantauan penggunaan memori selama proses berjalan dengan bantuan pustaka eksternal atau profiler

memori, sehingga informasi mengenai penggunaan memori bisa diketahui. Hasil pengukuran dari metode tersebut menjadi dasar untuk membandingkan efisiensi waktu dan penggunaan sumber daya tiga algoritma pengurutan yang diuji dalam setiap kategori dataset. Setelah semua pengujian selesai, hasilnya ditampilkan dalam bentuk tabel perbandingan. Tabel ini menunjukkan seberapa cepat dan efisien masing-masing algoritma dalam mengelola dataset skala kecil, menengah, dan besar.




**Gambar 1:** menunjukkan ringkasan skema yang dilakukan dalam penelitian..

## HASIL DAN PEMBAHASAN

Penelitian menggunakan bahasa pemrograman C++ dengan CodeBlocks sebagai lingkungan pengembangan terintegrasi (IDE) serta ekstensi Code Runner untuk menjalankan program. Untuk mengompilasi kode, kami menggunakan MinGW 64-bit agar program dapat berjalan dan di dokumentasikan hasilnya dengan Snipping Tool untuk mengambil gambar layar. Spesifikasi perangkat yang digunakan adalah: Sistem Operasi: Windows 11 (sistem operasi berbasis 64-bit, menggunakan prosesor x64)

1. RAM: 8 GB
2. CPU: AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx (2.10 GHz)
3. Penyimpanan: SSD 238 GB

Penelitian ini mengevaluasi cara kerja tiga algoritma pengurutan diantaranya: Bubble Sort, Selection Sort, dan Insertion Sort,. Ketiga algoritma diuji dengan menggunakan dataset yang sudah disediakan dalam ukuran standar dan dibagi menjadi tiga kelompok berdasarkan ukuran datanya, yaitu ArrayKecil yang berisi 100 elemen, Array Sedang yang memiliki 1.000 elemen, dan ArrayBesar yang terdiri dari 10.000 elemen. Data yang digunakan dihasilkan secara acak dengan angka-angka berupa bilangan bulat antara 1 sampai 99 agar pengujian tetap objektif.

Dalam proses pengujian, setiap algoritma dijalankan tiga kali di setiap kategori agar hasil yang diperoleh konsisten. Pada percobaan, dataset yang berisi 100 elemen diuji pada tes pertama hingga ketiga, dataset berisi 1.000 elemen diuji pada tes keempat hingga keenam, dan dataset yang berisi 10.000 elemen diuji pada tes ketujuh hingga kesembilan. Selama pengujian hanya dua aplikasi yang digunakan, yaitu CodeBlocks C++ Online Compiler untuk menjalankan kode dan Snipping Tools untuk membuat dokumentasi visual berupa screenshot. Hal ini dilakukan agar penggunaan CPU dan memori tidak terganggu. Kinerja diukur dengan mencatat durasi penghitungan dalam satuan nanodetik dan memantau penggunaan memori melalui metrik WorkingSetSize. Seluruh dataset serta laporan hasil pengujian masing-masing algoritma dapat diakses melalui tautan yang tersedia.

Dataset uji dapat dilihat di tautan berikut:

1. Bubble Sort https://bit.ly/48cIy3x
2. Selection Sort https://bit.ly/48nTdJP
3. Insertion Sort https://bit.ly/3JXZqD4

Bubble Sort

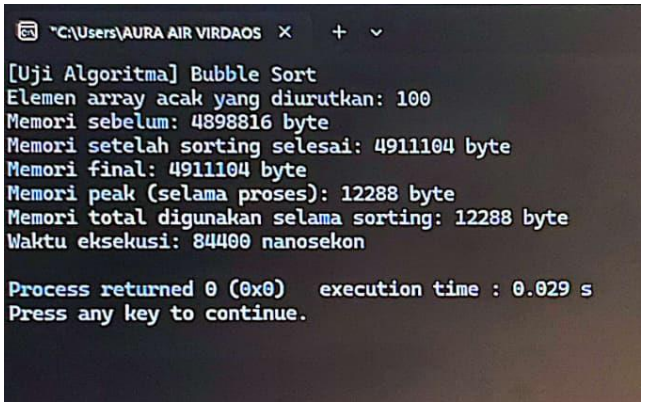
```
void bubbleSort(vector<int> &arr) {
bool swapped;
for (size_t i = 0; i < arr.size() - 1; i++) {
swapped = false;
for (size_t j = 0; j < arr.size() - i - 1; j++) {
if (arr[j] > arr[j + 1]) {
swap(arr[j], arr[j + 1]);
swapped = true;
}
}
if (!swapped)
break;
}
}
```

Dalam kode tersebut, algoritma Bubble Sort diaplikasikan menggunakan bahasa C++.Semua data uji sudah dimasukkan ke dalam sistem secara langsung dan dijalankan sesuai dengan skema pengujian yang sudah direncanakan sebelumnya. Tahap pengujian pertama hingga ketiga menggunakan 100 data, yaitu Dataset Kecil.. Tahap pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam menggunakan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan menggunakan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi serta penggunaan memori selama proses pengurutan dengan Bubble Sort dapat dilihat secara lengkap pada Tabel 1.

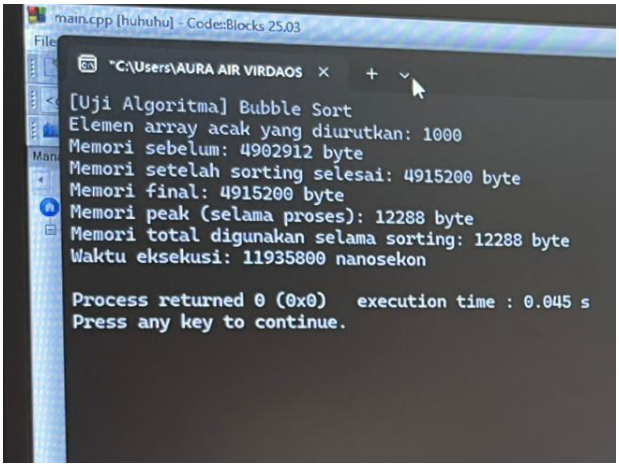
**Tabel 1: Hasil Uji Bubble sort**

| Percobaan ke | Ukuran data | Durasi eksekusi (nanodetik) | Konsumsi memori (byte) |
|---|---|---|---|
| 1 | 100 | 87000 | 12288 |
| 2 | 100 | 131300 | 12288 |
| 3 | 100 | 84400 | 12288 |
| 4 | 1.000 | 12176500 | 12288 |
| 5 | 1.000 | 11935800 | 12288 |
| 6 | 1.000 | 11992900 | 12288 |
| 7 | 10.000 | 1367502500 | 12288 |
| 8 | 10.000 | 1408719200 | 12288 |
| 9 | 10.000 | 1403459700 | 12288 |

Berdasarkan Tabel 1, rata-rata waktu yang dibutuhkan untuk mengurutkan data menggunakan Bubble Sort pada dataset dengan 100 data adalah 100.900 nanodetik. Ketika ukuran dataset diperbesar menjadi 1.000 data, waktu yang dibutuhkan meningkat menjadi 12.035.066,7 nanodetik, dan semakin bertambah besar hingga mencapai 1.393.227.133,333 nanodetik untuk 10.000 dataset. Sementara itu, penggunaan memori menampilkan pola yang tetap stabil, yaitu 12288 byte untuk 100 data, 12288 byte untuk 1.000 data, dan 12288 byte untuk 10.000 data.




**Gambar 2:** Keluaran Uji ke-3 (100 data) untuk Algoritma Bubble Sort




**Gambar 3:** Keluaran Uji ke-5 (1000 data) untuk Algoritma Bubble Sort Selection Sort

```
void selectionSort(int arr[], int n) {
   for(int i = 0; i < n - 1; i++) {
      int minIndex = i;
      for(int j = i + 1; j < n; j++) {
         if(arr[j] < arr[minIndex]) {
            minIndex = j;
         }
      }
      int temp = arr[i];
      arr[i] = arr[minIndex];
      arr[minIndex] = temp;
   }
}
```

Dalam kode tersebut, algoritma Selection Sort diaplikasikan menggunakan bahasa C++.Semua data uji sudah dimasukkan ke dalam sistem secara langsung dan dijalankan sesuai dengan skema pengujian yang sudah direncanakan sebelumnya. Tahap pengujian pertama hingga ketiga menggunakan 100 data, yaitu Dataset Kecil. Tahap pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam menggunakan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan menggunakan 10.000 data (Dataset Besar). Hasil pengukuran waktu

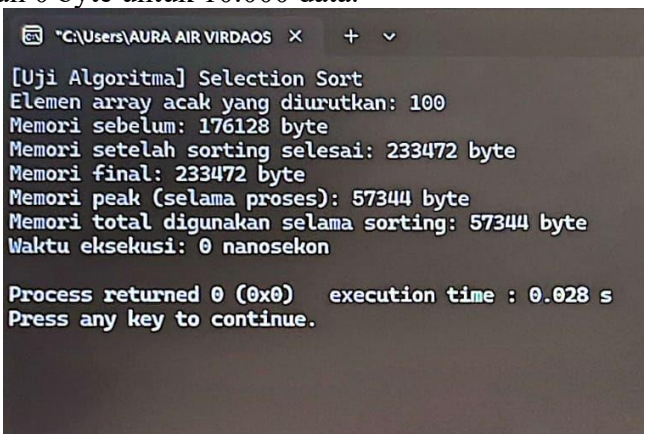
komputasi serta penggunaan memori selama proses pengurutan dengan Selection Sort dapat dilihat secara lengkap pada Tabel 2.

Percobaan ke Ukuran data Durasi eksekusi (nanodetik) Konsumsi memori (byte)

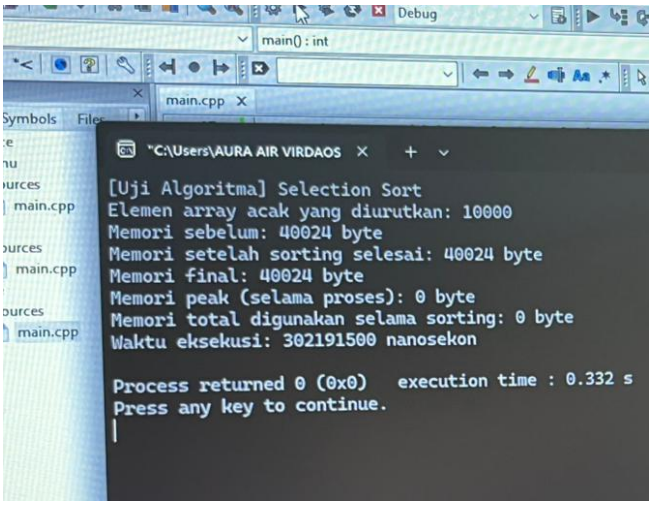
**Tabel 2: Hasil Uji Selection Sort**

| Percobaan ke | Ukuran data | Durasi eksekusi (nanodetik) | Konsumsi memori (byte) |
|---|---|---|---|
| 1 | 100 | 0 | 57344 |
| 2 | 100 | 0 | 57344 |
| 3 | 100 | 0 | 57344 |
| 4 | 1.000 | 1.000.000 | 61440 |
| 5 | 1.000 | 1.000.000 | 61440 |
| 6 | 1.000 | 2.000.000 | 61440 |
| 7 | 10.000 | 305.326.300 | 0 |
| 8 | 10.000 | 302.191.500 | 0 |
| 9 | 10.000 | 317.676.300 | 0 |

Berdasarkan Tabel 2, rata-rata waktu komputasi untuk Selection Sort pada dataset dengan 100 data adalah 0 nanodetik. Angka ini naik secara signifikan menjadi 1.333.333,33 nanodetik ketika jumlah data diperbesar menjadi 1.000, dan melonjak lagi menjadi 308.398.033 nanodetik untuk ukuran 10.000 dataset. Sementara itu, penggunaan memori menampilkan pola yang hampir tetap: 57344 byte untuk 100 data, 61440 byte untuk 1.000 data, dan 0 byte untuk 10.000 data.




**Gambar 4:** Keluaran Uji ke-3 (100data) untuk Algoritma Selection Sort




**Gambar 5:** Keluaran Uji ke-8 (1000data) untuk Algoritma Selection Sort

Insertion Sort

```
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;
   }
}
```

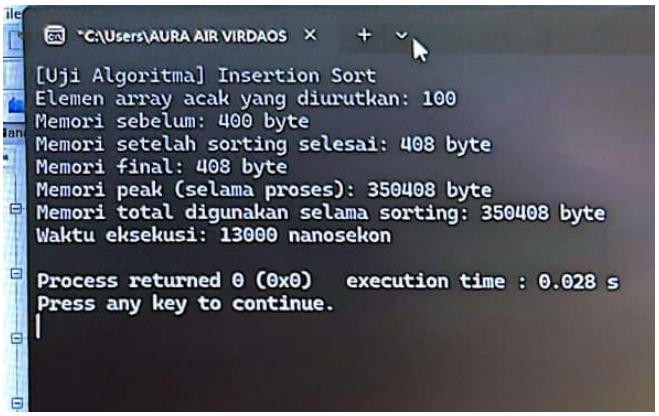
Dalam kode tersebut, algoritma Insertion Sort diaplikasikan menggunakan bahasa C++.Semua data uji sudah dimasukkan ke dalam sistem secara langsung dan dijalankan sesuai dengan skema pengujian yang sudah direncanakan sebelumnya. Tahap pengujian pertama hingga ketiga menggunakan 100 data, yaitu Dataset Kecil. Tahap pengujian pertama hingga ketiga menggunakan 100 data (Dataset Kecil), pengujian keempat hingga keenam menggunakan 1.000 data (Dataset Sedang), sedangkan pengujian ketujuh hingga kesembilan menggunakan 10.000 data (Dataset Besar). Hasil pengukuran waktu komputasi dan penggunaan memori selama proses pengurutan dengan Insertion Sort dapat dilihat secara lengkap pada Tabel 3.

Percobaan ke Ukuran data Waktu eksekusi (nanodetik) Konsumsi memori (byte)

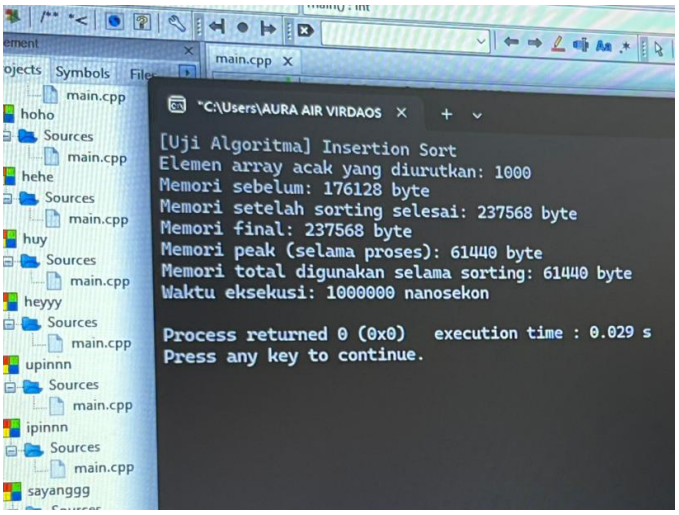
**Tabel 3: Hasil Uji Insertion Sort**

| Percobaan ke | Ukuran data | Waktu eksekusi (nanodetik) | Konsumsi memori (byte) |
|---|---|---|---|
| 1 | 100 | 10.700 | 350.408 |
| 2 | 100 | 13.000 | 350.408 |
| 3 | 100 | 10.400 | 350.408 |
| 4 | 1000 | 0 | 61.440 |
| 5 | 1000 | 1.000.000 | 61.440 |
| 6 | 1000 | 1.000.000 | 61.440 |
| 7 | 10000 | 90.000.000 | 98.304 |
| 8 | 10000 | 87.000.000 | 98.304 |
| 9 | 10000 | 89.000.000 | 98.304 |

Berdasarkan Tabel 3, rata-rata waktu komputasi untuk Insertion Sort pada data sebanyak 100 item adalah 11.366,6667 nanodetik. Angka ini meningkat besar menjadi 666.666,667 nanodetik ketika jumlah data ditingkatkan menjadi 1.000 item, dan melonjak lagi hingga 88.666.666,7 nanodetik ketika data mencapai 10.000 item. Sementara itu, penggunaan memori tetap stabil, yaitu 350.408 byte untuk 100 item, 61.440 byte untuk 1.000 item, dan 98.304 byte untuk 10.000 item.

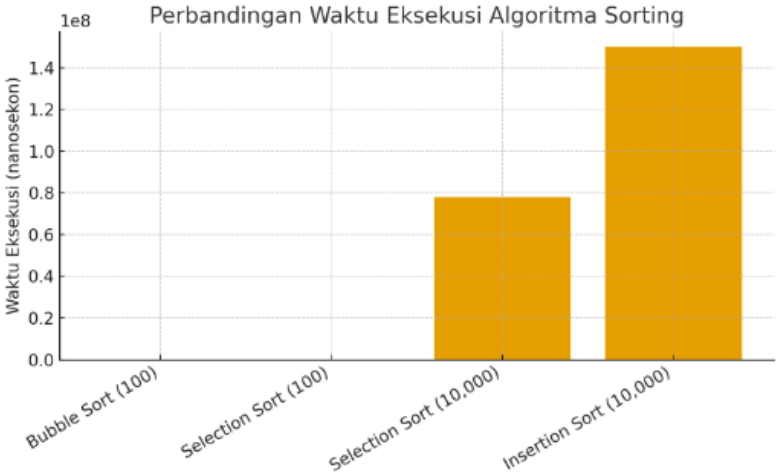



**Gambar 6:** Keluaran Uji ke-2(100 data) untuk Algoritma Insertion Sort

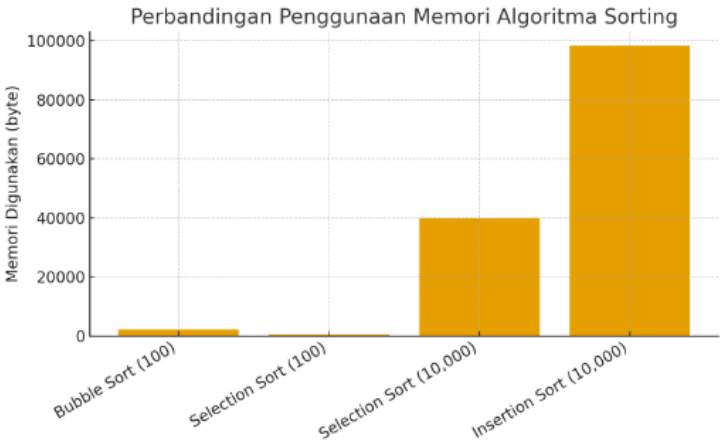



**Gambar 7:** Keluaran Uji ke-5 (1000 data) untuk Algoritma Insertion Sort

Berdasarkan hasil pengujian algoritma yang sudah dilakukan, gambar 8 dan gambar 9 di bawah ini menampilkan rata-rata hasil untuk setiap algoritma.




**Gambar 8:** Rata – Rata Waktu Eksekusi pada 3 Algoritma Pengurutan




**Gambar 9:** Rata – Rata Penggunaan Memori pada 3 Algoritma Pengurutan

Dari data pada Gambar 8 dan Gambar 9 yang menampilkan rata-rata ketiga metode pengurutan untuk tiga ukuran dataset yakni 100, 1.000, dan 10.000 elemen, dapat terlihat bahwa semakin besar jumlah data yang diproses, semakin lama waktu yang dibutuhkan untuk menjalankan algoritma tersebut.. Namun, penggunaan memori ketiganya tetap stabil karena ketiga algoritma ini termasuk dalam kategori pengurutan in-place.

Dari hasil rata-rata yang ditampilkan, Selection Sort terbukti paling efisien dalam penggunaan memori dan waktu ketika menangani data kecil, yaitu 100 elemen.

Bubble Sort sedikit lebih lambat dibandingkan Selection Sort, tetapi masih cukup stabil untuk data berukuran sedang. Sementara Insertion Sort membutuhkan waktu lebih lama ketika data meningkat sampai 10.000 elemen, meskipun performanya tetap bagus untuk data kecil atau yang sudah hampir terurut.

Secara keseluruhan, Selection Sort merupakan algoritma yang seimbang dalam hal kecepatan dan penggunaan memori, Insertion Sort kurang efisien saat mengolah data besar, sedangkan Bubble Sort berada di tengah antara keduanya.

## KESIMPULAN

Berdasarkan hasil pengujian dan analisis terhadap tiga metode pengurutan, yakni Bubble Sort, Selection Sort, dan Insertion Sort, dapat disimpulkan bahwa ukuran data sangat memengaruhi waktu yang dibutuhkan dan penggunaan memori masing-masing metode. Semakin besar jumlah data yang diproses, semakin lama waktu eksekusinya karena ketiga algoritma ini memiliki kompleksitas waktu sebesar $O(n^2)$. Hal ini terlihat jelas ketika jumlah elemen meningkat dari 100 menjadi 10.000, di mana waktu pemrosesan bisa bertambah ratusan hingga ribuan kali lipat. Secara umum, Selection Sort menunjukkan performa yang paling stabil dan cepat dibandingkan kedua algoritma lainnya.

Dalam data berukuran kecil, Selection Sort dapat menyelesaikan pengurutan dengan waktu yang tidak terlalu lama dan memakai memori yang tidak banyak. Bubble Sort memiliki waktu eksekusi sedikit lebih lama dibanding Selection Sort, namun tetap dalam batas yang wajar. Sementara itu, Insertion Sort memiliki waktu eksekusi terlama terutama ketika data besar, karena jumlah perbandingan dan pergeseran elemen meningkat secara signifikan. Namun, untuk data yang sudah hampir terurut, Insertion Sort bisa bekerja lebih cepat dibanding dua algoritma lainnya.

Dari sisi penggunaan memori, ketiga algoritma tidak terlalu berbeda. Hal ini dikarenakan ketiganya termasuk dalam kategori algoritma in-place sorting, yaitu pengurutan yang dilakukan langsung di dalam array tanpa membutuhkan memori tambahan yang besar. Meski begitu, peningkatan ukuran data tetap menyebabkan sedikit kenaikan penggunaan memori akibat kebutuhan ruang sementara saat proses perbandingan dan

pertukaran elemen berlangsung.

Secara keseluruhan, hasil analisis menunjukkan bahwa Selection Sort merupakan algoritma yang paling optimal secara rata-rata karena mampu memberikan keseimbangan antara kecepatan dan efisiensi penggunaan memori. Bubble Sort cocok digunakan untuk pengurutan sederhana dengan ukuran data tidak terlalu besar, sedangkan Insertion Sort lebih tepat digunakan pada data yang sudah mendekati terurut. Dengan memahami karakteristik dan performa masing-masing algoritma, pengguna bisa memilih metode pengurutan yang paling sesuai dengan kebutuhan dan kondisi data yang dihadapi.

## DAFTAR PUSTAKA


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++.

Arifin, R. W., & Setiyadi, D. (2020). Algoritma Metode Pengurutan Bubble Short dan Quick Sort Dalam Bahasa Pemprogaman c++. Information System For Educators and Profesionals, 2(No.4), 178–187.

Frühwirth, T. (2010). Union-find algorithm. In Constraint Handling Rules. https://doi.org/10.1017/cbo9780511609886.014

Knuth, D. E. (1998). Art of Computer Programming - Volume 3 (Sorting & Searching) (Second Edi). Addison-Wesley Professional.

Munir, R., & Lidya, L. (2016). Algoritma dan Pemrograman dalam Bahasa Pascal, C, Dan C++ Edisi Keenam. Informatika Bandung.

Poetra, D. R. (2022). Performa Algoritma Bubble Sort dan Quick Sort pada Framework Flutter dan Dart SDK(Studi Kasus Aplikasi E-Commerce). JATISI (Jurnal Teknik Informatika Dan Sistem Informasi), 9(2), 806–816. https://doi.org/10.35957/jatisi.v9i2.1886

Pratama, A., Desiani, A., & Irmeilyana, I. (2018). Analisis Kebutuhan Waktu Algoritma INsertion Sort, Merge Sort, dab Quick Sort dengan Kompleksitas Waktu. In Computer Science and ICT, Vol 2(No. 1), 95-1–6.

Pujiono, I. P., Rachmawanto, E. H., Anisa, N., & Winarsih, S. (2025). Array Sorting Algorithm vs Algoritma Pengurutan Tradisional : Analisis Efisiensi Memori dan Waktu Array Sorting Algorithm vs Traditional Sorting Algorithm : Memory and Time Efficiency Analysis. 15(April), 47–59.

Pujiono, I. P., Trianto, R. B., & Hana, F. M. (2024). Perbandingan Efisiensi Memori dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java. Simkom, 9(2), 218–230. https://doi.org/10.51717/simkom.v9i2.481

Rahayuningsih, P. (2016). Analisis Perbandingan Kompleksitas Algoritma Pengurangan NIlai (Sorting). Jurnal Evolusi, No. 4, 64–75.

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, Vol. 2(No. 2), 136–145.

Syafnidawaty. (2020). Pengertian Logika Fuzzy. 06 April 2020, II(September), 3.

Ullah, Z. (2016). Understanding Sorting Techniques Using C++. International Journal of Novel Research in Computer Science and Software Engineering, 3(1), 171–199.

Yagci, Y., & Mishra, M. K. (2016). Data and structures. In Handbook of Vinyl Polymers: Radical Polymerization, Process, and https://doi.org/10.1201/9781420015133.pt5 Technology: Second Edition.

Pujiono, I. P., Rikzam Kamal, M., Prayogi, A., Sari, C. A., & Ikhsanuddin, R. M. (2024).

Perbandingan Efisiensi Memori dan Waktu Komputasi pada 7 Algoritma Sorting (Bubble, Insertion, Selection, Shell, Quick, Merge, Heap). SIMKOM — E-Jurnal STMIK Bina Sarana Informatika

Pujiono, I. P., Kamal, M. R., Prayogi, A., Atika Sari, C., & Ikhsanuddin, R. M. (2025). Analisis Efisiensi Memori dan Waktu Komputasi: Counting Sort vs Algoritma Pengurutan Modern. JITET (Journal of Information Technology Engineering & Technology). DOI: 10.23960/jitet.v13i3.6657.

Musyaffa, M. Z., Raharjo, K., Faiz, M., Ammarullah, S., & Pujiono, I. P. (2025). Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algoritma Pengurutan Tradisional Menggunakan Python. JEIS / ejurnal Swadharma (PDF).

Basir, R. R. (2020). Analisis Kompleksitas Ruang dan Waktu terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort, dan Merge (artikel / PDF). Jurnal STRING — LPPM Universitas Indraprasta (Unindra).

Retnoningsih, E. (tahun tidak tercantum). Algoritma Pengurutan Data (Sorting) — studi komparatif Selection Sort vs Insertion Sort (laporan/ makalah). Neliti (PDF).

Hoare, C. A. R. (1962). Quicksort. The Computer Journal, 5(1), 10–16. (Karya asli yang memperkenalkan Quicksort — sering dikutip pada studi komparatif).

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). The MIT Press. (Buku rujukan utama tentang kompleksitas waktu/ruang algoritma).