JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

ANALISIS PERBANDINGAN EFISIENSI ALGORITMA *INTROSORT* DENGAN ALGORITMA TRADISIONAL *BUBBLE SORT* DAN *SELECTION SORT*

Salsabila Amanda[1)], Lilina Dwi Anastasya[2)], Fira Sulistia[3)], Nafisa Purnamasari[4)], Imam Prayogo Pujiono[5)]
[1,2,3,4]Prodi Bisnis Digital, Fakultas Ekonomi dan Bisnis Islam, Universitas Islam Negeri K.H. Abdurrahman Wahid Pekalongan
[5]Prodi Informatika, Fakultas Ekonomi dan Bisnis Islam, Universitas Islam Negeri K.H. Abdurrahman Wahid Pekalongan

Correspondence author: S. Amanda, salsabila.amanda25002@mhs.uingusdur.ac.id, Pekalongan, Indonesia

Abstract

Data sorting is a fundamental operation that significantly impacts the performance of various computing applications. In C++, developers can choose from traditional sorting algorithms, such as Bubble sort and Selection sort, as well as modern algorithms, such as Introsort. This study aims to analyze and compare the efficiency of the Introsort, Bubble sort, and Selection sort algorithms in terms of execution time and memory usage. The method used is a comparative experiment with three random data sizes: 100, 1,000, and 10,000 elements. Each scenario is tested three times using a recursive C++ implementation on a Lenovo Ideapad 1i laptop with an Intel Celeron N4020 processor, 8 GB of RAM, and a 477 GB SSD. Execution time is measured using the std::chrono library, while memory usage is measured using the getMemoryUsage() function. The results show that Introsort is consistently the fastest algorithm for all data sizes. At 10,000 elements, the average execution time of Introsort is about 2.88 ms, while Bubble sort and Selection sort are about 1,303 ms and 1,067 ms, respectively. Thus, Introsort is about 450 times faster than Bubble sort and 370 times faster than Selection sort on large datasets, while the difference in memory usage among the three algorithms is under 0.15 MB. These findings indicate that choosing a modern algorithm like Introsort is highly recommended for sorting large data sets in C++. In contrast, traditional algorithms are more appropriate for learning scenarios or small datasets.

Keywords: comparison, sorting algorithm, Introsort, bubble sort, selection sort

Abstrak

Pengurutan data merupakan operasi dasar yang sangat memengaruhi kinerja berbagai aplikasi komputasi. Pada bahasa pemrograman C++, pengembang dapat memilih algoritma pengurutan tradisional seperti *Bubble sort* dan *Selection sort*, maupun algoritma modern seperti *Introsort*. Penelitian ini bertujuan menganalisis dan membandingkan efisiensi algoritma *Introsort*, *Bubble sort*, dan *Selection sort* berdasarkan waktu eksekusi dan penggunaan memori. Metode yang digunakan adalah eksperimen komparatif dengan tiga skala ukuran data acak, yaitu 100, 1.000, dan 10.000 elemen. Setiap skenario diuji sebanyak tiga kali menggunakan implementasi rekursif C++ pada laptop Lenovo Ideapad 1i dengan prosesor Intel

Celeron N4020, RAM 8 GB, dan SSD 477 GB. Waktu eksekusi diukur menggunakan pustaka *std::chrono*, sedangkan penggunaan memori diukur dengan fungsi *getMemoryUsage()*. Hasil menunjukkan bahwa *Introsort* secara konsisten menjadi algoritma paling cepat untuk seluruh ukuran data. Pada 10.000 elemen, rata-rata waktu eksekusi *Introsort* sekitar 2,88 ms, sedangkan *Bubble sort* dan *Selection sort* masing-masing sekitar 1.303 ms dan 1.067 ms. Dengan demikian, *Introsort* sekitar 450 kali lebih cepat daripada *Bubble sort* dan 370 kali lebih cepat daripada *Selection sort* pada skala data besar, sementara perbedaan penggunaan memori di antara ketiga algoritma berada di bawah 0,15 MB. Temuan ini mengindikasikan bahwa pemilihan algoritma modern seperti *Introsort* sangat disarankan untuk pengurutan data berukuran besar di C++, sedangkan algoritma tradisional lebih tepat digunakan untuk skenario pembelajaran atau dataset kecil.

Kata Kunci: perbandingan, algoritma pengurutan, *Introsort*, *bubble sort*, *selection sort*

A. PENDAHULUAN

Menurut Goodman Hedet Niemi, algoritma adalah langkah-langkah terbatas yang memerlukan memori dan waktu untuk menyelesaikan suatu permasalahan (Aulia & Yahfizham, 2024). Program merupakan kumpulan instruksi yang dinyatakan dalam bentuk kode atau format lain. Langkah-langkah ini jika digabung dengan media yang dapat diakses oleh komputer, dapat menyebabkan komputer menjalankan fungsi-fungsi khusus, seperti persiapan dan perencanaan langkah-langkah tersebut (Dharmalau et al., 2025). Dapat disimpulkan bahwa algoritma merupakan instruksi sistematis, sedangkan pemrograman merupakan tindakan dalam pembuatan program. Sehingga algoritma pemrograman adalah instruksi sistematis untuk membuat suatu program menggunakan bahasa pemrograman untuk memecahkan masalah dan mencapai tujuan tertentu (Murad et al., 2025). Didefinisikan juga oleh (Yanti & Eriana, 2024) algoritma pemrograman adalah langkah sistematis dalam membuat program yang menggunakan bahasa pemrograman untuk memecahkan suatu masalah.

Sorting merupakan proses penyusunan urutan dari sejumlah data (Retnoningsih, 2018). Pengurutan data bisa dilakukan secara *ascending* (menaik) dan *descending* (menurun) (Sari et al., 2022). Tujuan utama pengurutan data adalah untuk menganalisis, mempermudah pencarian, yang sering kali diterapkan dalam algoritma komputer. Algoritma *sorting* merupakan pengurutan data acak yang dilakukan dengan bahasa pemrograman tertentu. Dalam bahasa pemrograman C++ ada dua macam pengurutan data, yaitu *sorting* tradisional dan *sorting* modern. *Sorting* tradisional diantaranya adalah *Bubble sort*, *Selection sort*, dan *Insertion Sort*. Perbedaan *sorting* tradisional dan modern adalah keefisiensian dalam penyusunan data secara teratur dalam jumlah besar.

Saat ini ada banyak macam metode pengurutan data yang digunakan, untuk membatasi luas dari pembahasan ini penulis hanya akan membahas tiga pengurutan data, dari algoritma terbaru diambil sampel algoritma *Introsort* dengan algoritma tradisional, yaitu *Bubble sort* dan *Selection sort*. Tujuan penelitian ini adalah untuk mengetahui perbandingan dari ketiga algoritma tersebut yang mana nantinya akan memperlihatkan secara jelas kinerja dari segi kecepatan dan juga akurasi dalam pengurutan data sesuai dengan panjang data dari masing-masing iterasi. *Introsort* adalah algoritma hibrid yang merupakan penggabungan dari tiga algoritma *sorting* yaitu *Heap Sort*, *Quick Sort*, dan *Insertion Sort*. Proses eksekusi *Introsort* dimulai seperti *median of three Quick Sort* dan nantinya akan berubah ke

sistematis *Heap Sort* jika ditemukan penurunan kinerja *sorting*. *Introsort* memiliki kecepatan yang mampu melebihi *Quick Sort* pada input berbahaya. Namun, lebih cepat dari *Heap Sort* pada data yang lebih acak (Musser, 1997).

Algoritma *Bubble sort* merupakan algoritma yang terinspirasi dari perilaku gelembung sabun di atas permukaan air. Algoritma *Bubble sort* adalah algoritma yang bekerja dengan membandingkan data yang berdekatan dalam rentang lalu menukar posisinya jika tidak dalam urutan yang tepat (Rachmat, 2018). Proses pengurutan ini diulangi sampai seluruh data terurut. Alur *Bubble sort* dimulai dengan membandingkan setiap pasangan data kemudian dilakukan pertukaran jika ditemukan data yang acak atau tidak sesuai dengan urutan kemudian diulangi sampai tidak diperlukan lagi pertukaran data (Aulia & Yahfizham, 2024).

Selection sort adalah algoritma yang sederhana dan dilakukan pass beberapa kali untuk melakukan seleksi pada data. Pada pengurutan *ascending* (naik) algoritma memerlukan data dengan nilai terkecil dari bagian data yang belum terurut, kemudian mencatat indeksnya dan menukar data tersebut dengan data pada posisi paling depan yang belum tersusun. Untuk pengurutan *descending* (turun), yang dicari adalah data terbesar, lalu indeksnya diatur dengan posisi awal bagian yang belum terurut. Algoritma ini dikenal karena kesederhanaannya, dan dalam kondisi tertentu kinerjanya bisa lebih baik dibandingkan beberapa algoritma yang lebih kompleks (Yovita et al., 2023).

Dari berbagai implementasi algoritma, *Sorting* atau pengurutan data merupakan suatu program yang penting dan bertujuan untuk menyusun dari data acak, memudahkan pencarian, analisis, dan pengelolaan data. Beberapa penelitian sudah melakukan analisis perbandingan dari *Bubble sort* dan *Selection sort*, namun masih jarang peneliti menganalisis tentang algoritma *Introsort*. Berikut beberapa penelitian yang melakukan analisis terhadap *Bubble sort* dan *Selection sort*. Dari penelitian (Mahrozi & Faisal, 2023) membandingkan kecepatan dari algoritma *Bubble sort* dan *Selection sort*, yang menemukan bahwa *Selection sort* lebih cepat dibandingkan *Bubble sort* dalam pengeksekusiannya pada ukuran data 100 sampai dengan 100.000. Kemudian menurut (Gustina et al., 2025) perbedaan kecepatan antar algoritma pengurutan sangat dipengaruhi oleh kondisi data (acak, hampir terurut, atau terurut) dan cara penerapannya dalam perangkat lunak. Penelitian milik (Arifin & Setiyadi, 2020), membandingkan antara algoritma *Bubble sort* dan *Quick Sort*, dihasilkan bahwa algoritma *Bubble sort* membandingkan setiap elemen pada array dengan maksimal, terbukti dari performanya yang melakukan 7 kali putaran pada array berjumlah sepuluh dan melakukan 10 kali perbandingan antara n dan n-1. Namun juga, algoritma ini menggunakan memori yang cukup besar, sehingga kurang efisien untuk data berukuran besar. Penelitian milik (Zahwa et al., 2025), membandingkan antara algoritma-algoritma *sorting* tradisional, menjelaskan bahwa algoritma sederhana seperti *Bubble sort* dan *Selection sort* memang menggunakan kapasitas memori yang cukup stabil, namun performanya buruk untuk pengurutan pada array besar. Penelitian milik (Pujiono et al., 2024), membandingkan antara 7 algoritma *sorting*, mendapatkan hasil bahwa *Bubble sort* secara konsisten merupakan algoritma dengan kecepatan eksekusi lambat, dan lebih disarankan menggunakan *Heap Sort* untuk data 10.000.

Penelitian milik (Riki et al., 2024) tentang Perbandingan *Selection sort*, Shell Sort, Dan Merge Sort. Penelitian (Marcellino et al., 2021) tentang komparasi antara *Quick Sort*, *Heap Sort*, Merge Sort, *Introsort*, dan Radix Sort atas dasar waktu dan pemakaian memori, mendapatkan bahwa Introspective Sort (*Introsort*) merupakan algoritma paling cepat, dan *Heap Sort* merupakan algoritma dengan penggunaan kapasitas memori paling minimum.

Dengan demikian, masih terdapat ruang penelitian terkait pengujian yang secara sistematis membandingkan algoritma *sorting*

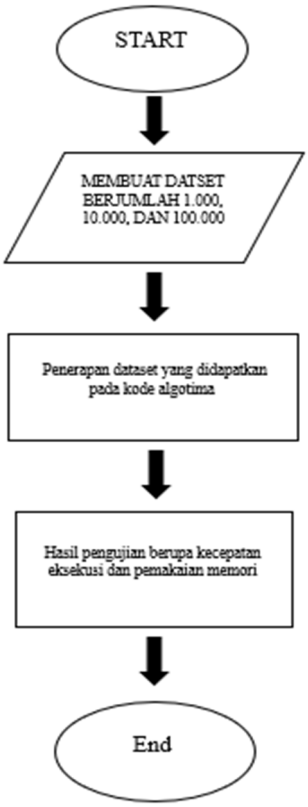
modern seperti *Introsort* dengan algoritma tradisional *Bubble sort* dan *Selection sort* dalam lingkungan bahasa C++. Terutama, belum banyak kajian yang secara bersamaan mengevaluasi waktu eksekusi dan penggunaan memori pada berbagai skala data acak. Kelemahan penelitian sebelumnya ada pada dataset yang dipakai, karena dataset yang dipakai relatif berskala kecil hasilnya kurang mewakili aplikasi dengan data besar dan kompleks (Zulfa et al., 2022). Penelitian lain yang menguji *Selection sort* (Yunial, 2025) mengatakan bahwa *Selection sort* memiliki kecepatan dan performa lebih baik dari *Bubble sort*. Namun, *Selection sort* memiliki skalabilitas terbatas karena kompleksitasnya sekitar O(n2), sehingga kurang cocok juga dengan dataset besar. Studi eksperimen yang dilakukan oleh peneliti kompleksitas (Dhamma, 2024) melampirkan bahwa analisis kompleksitas pada *Selection sort* ini efisiensinya tidak berubah dan tetap pada kompleksitas O(n2) pada berbagai skenario dataset yang digunakan.

Secara spesifik, penelitian ini menawarkan tiga kontribusi utama. (1) Menyajikan implementasi rekursif *Introsort*, *Bubble sort*, dan *Selection sort* dalam bahasa pemrograman C++ dengan lingkungan pengujian yang terdokumentasi secara rinci. (2) Melakukan evaluasi komparatif berdasarkan dua metrik, yaitu waktu eksekusi dan penggunaan memori, pada tiga skala ukuran data acak (100, 1.000, dan 10.000 elemen). (3) Memberikan rekomendasi praktis pemilihan algoritma pengurutan untuk skenario array kecil hingga besar, sehingga dapat menjadi rujukan awal bagi pengembang dan peneliti saat memilih algoritma *sorting* yang efisien di C++.

B. METODE PENELITIAN

Penelitian ini menggunakan metode komparatif. Tujuan penelitian ini adalah untuk membandingkan kecepatan *Introsort* dengan dua algoritma tradisional, yaitu *Bubble sort* dan *Selection sort* berdasarkan kecepatan dan efisiensi proses pengurutan data. Melalui pendekatan komparatif, penelitian ini bertujuan untuk mengetahui algoritma mana yang paling efisien antara algoritma *sorting* hibrid terbaru *Introsort* dengan algoritma pengurutan data tradisional *Bubble sort* dan *Selection sort*. Penelitian ini berfokus pada pemrograman C++ karena pengendalian pada tingkat yang lebih rendah terhadap pemakaian kapasitas memori dan waktu eksekusi.

Penelitian ini memanfaatkan bahasa pemrograman C++ dan *Code::Blocks* versi 25.03-r13644 sebagai sarana pendukung. Seluruh pengujian algoritma ini menggunakan laptop Lenovo Ideapad 1i (14",7) dengan spesifikasi Processor Intel(R) Celeron(R) N4020 CPU dengan kecepatan 1.10GHz, RAM 8,00 GB, System Type 64-bit operating system, dan SSD 477GB.



Gambar 1. *Flowchart* Pengujian

Penelitian ini diawali dengan mencetak dataset berupa kumpulan angka acak dengan banyak data yang sudah ditentukan yaitu data 100 (array kecil), 1.000 (array sedang), dan 10.000 (array besar) yang akan didapatkan

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

dari aplikasi *Number Generator* (https://www.calculator.net/). Kemudian dataset yang telah didapatkan diterapkan pada code program C++ yang telah dibuat. Code program C++ ini akan dijalankan pada aplikasi Code::Blocks versi 25.03-r13644 dengan metode rekursif. Hasil pengujian yang tercetak kemudian dihitung rata-ratanya dan dikomparasikan.

C. HASIL DAN PEMBAHASAN

Hasil Penelitian

Pengujian algoritma ini meneliti antara algoritma terbaru *Introsort* dengan algoritma tradisional *Bubble sort* dan *Selection sort*. Dari ketiga algoritma tersebut mengunakan dataset yang sama, yaitu terdiri dari 100 data (array kecil), 1.000 data (array sedang), dan 10.000 data (array besar). Data yang digunakan berisi angka acak yang diperoleh dari aplikasi Number Generator.

Step pengujian dilakukan sebanyak tiga kali setiap jumlah data set. Hasil pengujian bisa diakses melalui link berikut :

1. File Dataset (bit.ly/44lwMm5)
2. File Hasil Pengujian *Bubble sort* (bit.ly/488gtKM)
3. File Hasil Pengujian *Selection sort* (bit.ly/4a2y1KE)
4. Hasil Pengujian *Introsort* (bit.ly/482AXUX).

Step pertama, dilakukan pengujian dengan memasukkan dataset berjumlah 100 ke dalam algoritma, kemudian dihitung kecepatan eksekusi mengunakan rumus std::chrono:: dan penggunaan kapasitas memori mengunakan rumus getMemoryUsage(). Step kedua dan ketiga dilakukan sama persis seperti step pertama, dengan memasukkan dataset berjumlah 1.000 dan 10.000 sebanyak tiga kali, lalu setelah data tercetak, hasil akan dihitung untuk pencarian rata-rata. Step terakhir adalah meng-compare hasil dari ketiga algoritma. Berikut adalah penjelasan setiap algoritma secara rinci.

Bubble sort

Program :

```
void bubbleSortRec(std::vector<int>& arr, int
n) {
   if (n == 1) return;
   for (int i = 0; i < n - 1; i++) {
      if (arr[i] > arr[i + 1]) {
         std::swap(arr[i], arr[i + 1]);
      }
   }
   bubbleSortRec(arr, n - 1);
}
```

Algoritma *Bubble sort* rekursif (*ascending*) bekerja dengan cara mengurutkan data secara bertahap lewat proses rekursi. Program melakukan pengecekan dari awal array hingga data sebelum akhir. Dua data yang berdampingan akan saling dibandingkan, kemudian ditukar urutannya jika salah (elemen di kiri lebih besar) (Nasution et al., 2023). Setelah satu siklus pengecekan selesai, bilangan terbesar sudah otomatis berada di paling akhir array. Lalu kemudian dilanjutkan dengan memanggil kembali fungsi yanng sebelumnya namun dengan ukuran array yang sudah dikurangi satu, sebagai artian bahwa elemen yang berada di akhir array merupakan elemen terbesar dan sudah benar urutannya.

Tabel 1. Hasil Pengujian *Bubble sort*

Data	Waktu (ms)	Memori (MB)
100	0.7020	4.4531
100	0.2482	4.0156
100	0.2541	3.9883
1.000	10.4101	4.4570
1.000	40.0284	4.0156
1.000	32.5601	4.0078
10.000	1132.0632	4.0547
10.000	1729.9797	4.0547
10.000	1047.4152	4.0273

Dapat dilihat pada tabel bahwa algoritma *Bubble sort* ini berjalan konsisten dengan lambat, sehingga algoritma *Bubble sort* hanya akan efektif apabila digunakan pada array kecil. Hal ini sejalan dengan penelitian (Ali et al., 2025) yang menyatakan bahwa *Bubble sort* konsisten dengan kelambatannya.

Selection sort

Program :

```
void selectionSortRec(std::vector<int>& arr, int n, int start = 0) {
   if (start >= n - 1) return;

   int minIndex = start;
   for (int i = start + 1; i < n; ++i) {
      if (arr[i] < arr[minIndex]) {
         minIndex = i;
      }
   }
   std::swap(arr[start], arr[minIndex]);
   selectionSortRec(arr, n, start + 1);
}
```

Algoritma *Selection sort* rekursif bekerja dengan cara menelusuri elemen dengan nilai paling kecil dari urutan array, kemudian ditempatkan pada posisi yang tepat. Pada tiap langkah pengurutannya, algoritma mencari elemen dengan nilai terkecil dan ditukar dengan posisi terdepan, setelah posisi awal terisi dengan benar, fungsi Selection akan berjalan mengurutkan sisa elemen lain pada array hingga urutan pada array benar. Algoritma di atas mengurutkan array dari nilai terkecil hingga terbesar (*ascending*). Pendekatan rekursif membuat proses berlangsung hingga array terurut seluruhnya.

Tabel 2. Hasil Pengujian *Selection sort*

Data	Waktu (ms)	Memori (MB)
100	0.2622	3.9961
100	0.4530	4.0117
100	2.3010	4.0117
1.000	16.0294	4.0078
1.000	11.4190	4.0195
1.000	18.3960	4.0078
10.000	1066.7334	4.0234
10.000	1052.4875	4.0430
10.000	1081.7970	4.0430

Dapat dilihat pada tabel, bahwa *Selection sort* cukup stabil dalam pengeksekusian dan pemakaian memori, namun *Selection sort* tidak terlalu efisien untuk pengurutan data besar. Sama seperti halnya pada penelitian (Tambunan et al., 2024), bahwa *Selection sort* akan stabil pada data berukuran besar namun bergerak dengan lambat.

Introsort

Program :

Fungsi *Insertion Sort* untuk array kecil

```
void insertionSort(std::vector<int>& arr, int low, int high) {
   for (int i = low + 1; i <= high; ++i) {
      int key = arr[i];
      int j = i - 1;
      while (j >= low && arr[j] > key) {
         arr[j + 1] = arr[j];
         --j;
      }
      arr[j + 1] = key;
   }
}
```

Fungsi Heapify untuk *Heap Sort*

```
void heapify(std::vector<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) {
      std::swap(arr[i], arr[largest]);
      heapify(arr, n, largest);
   }
}
```

Fungsi *Heap Sort*

```
void heapSort(std::vector<int>& arr, int low, int high) {
   int n = high - low + 1;
   for (int i = n / 2 - 1; i >= 0; --i) {
      heapify(arr, n, i);
   }
   for (int i = n - 1; i > 0; --i) {
      std::swap(arr[0], arr[i]);
      heapify(arr, i, 0);
   }
}
```

Fungsi Partition untuk *Quick Sort*

```
int partition(std::vector<int>& arr, int low, int high) {
   int pivot = arr[high];
```

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 01 JANUARI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

```
    int i = low - 1;
    for (int j = low; j < high; ++j) {
        if (arr[j] < pivot) {
            ++i;
            std::swap(arr[i], arr[j]);
        }
    }
    std::swap(arr[i + 1], arr[high]);
    return i + 1;
}
```

Fungsi *Introsort* Hybrid

```
void IntrosortUtil(std::vector<int>& arr, int low, int high, int depthLimit) {
    int n = high - low + 1;
    if (n <= 16) {  // Threshold untuk Insertion Sort
        insertionSort(arr, low, high);
        return;
    }
    if (depthLimit == 0) {
```

Jika kedalaman terlalu dalam, gunakan *Heap Sort*

```
        heapSort(arr, low, high);
        return;
    }
```

Partition *Quick Sort*

```
    int pivot = partition(arr, low, high);
    IntrosortUtil(arr, low, pivot - 1, depthLimit - 1);
    IntrosortUtil(arr, pivot + 1, high, depthLimit - 1);
}
```

Fungsi utama *Introsort*

```
void Introsort(std::vector<int>& arr) {
    int n = arr.size();
    int depthLimit = 2 * static_cast<int>(log(n));  // Kedalaman maksimal
    IntrosortUtil(arr, 0, n - 1, depthLimit);
}
```

Introsort adalah algoritma yang menggabungkan tiga kecepatan yaitu *Quick Sort*, *Insertion Sort*, dan *Heap Sort*. Algoritma ini bekerja secara rekursif dengan mengunakan fungsi *Introsort*Util, yang membagi array menjadi dua bagian seperti pada *Quick Sort*. Algoritma akan otomatis berpindah ke fungsi *Insertion Sort* jika menghadapi sub-array yang kecil untuk mengoptimalkan kinerja. Jika kedalaman rekursif sudah mencapai limit, algoritmanya akan berpindah dari *Quick Sort* ke *Heap Sort* untuk menjaga notasi kecepatan waktu eksekusi. *Introsort* mengunakan kombinasi ketiga algoritma tersebut untuk mempertahankan kinerja dan membuat strategi paling tepat apabila menghadapi skenario array tertentu.

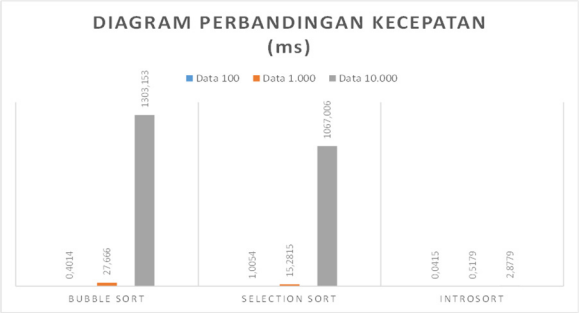
Tabel 3. Hasil Pengujian *Introsort*

Data	Waktu (ms)	Memori (MB)
100	0.0448	4.0156
100	0.0235	4.0117
100	0.0521	4.0156
1.000	0.5109	3.9922
1.000	0.5249	4.4453
1.000	0.5179	4.0078
10.000	3.0050	4.0469
10.000	3.0120	4.0547
10.000	2.6167	4.0430

Dapat dilihat pada data bahwa *Introsort* memiliki kecepatan eksekusi yanng sangat cepat dan penggunaan memorinya masih cukup stabil untuk data besar sekalipun. Data tersebut sesuai dengan penelitian milik David R. Musser selaku penemu algoritma *Introsort* sendiri, bahwa *Introsort* memiliki operasi yang lebih efisien, algoritma ini menggunakan penggabungan dari *Quick Sort* dan *Heap Sort* untuk menjaga performa (Musser, 1997).

Perbandingan Kecepatan Ketiga Algoritma

Hasil perbandingan diambil dari rata-rata tiga percobaan yang telah dilakukan. Diagram tersebut memperlihatkan data dan waktu komparasi yang berbeda. Algoritma *Bubble sort* memiliki kecepatan paling lama dibandingkan dengan algoritma lainnya. Performa *Bubble sort* hanya akan efisien untuk array kecil (100) kurang efisien untuk array sedang (1.000) dan besar (10.000).



Gambar 1. Grafik Perbandingan Kecepatan

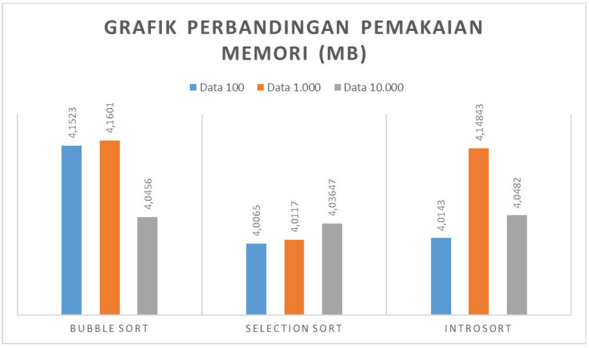
Tabel 4. Rata-rata kecepatan keseluruhan percobaan

Jumlah Data	Rata-rata Kecepatan (ms)		
	Bubble sort	***Selection sort***	***Introsort***
100	0,4014	1,0054	0,0415
1.000	27,666	15,2815	0,5179
10.000	1303,153	1067,006	2,8779

Jika dibandingkan antara *Bubble sort* dengan *Selection sort*, hasil menujukan bahwa *Selection sort* lebih unggul dari aspek kecepatan dibandingkan algoritma *Bubble sort*. Di sisi lain *Selection sort* tidak lebih unggul dibandingkan algoritma *Introsort*. Waktu eksekusi *Introsort* pada array besar bahkan tidak menyentuh 5ms, bisa dilihat bahwa algoritma ini memiliki kecepatan eksekusi yang luar biasa cepat. Bisa disimpulkan, bahwa waktu eksekusi *Introsort* lebih efisien baik untuk array kecil maupun array besar dibanding *Bubble sort* dan *Selection sort*.

Perbandingan Penggunaan Memori Ketiga Algoritma

Perbandingan diagram tersebut didapatkan dari hasil rata-rata ketiga percobaan sebelumnya yang telah dilakukan. Setiap algoritma memperlihatkan hasil ukuran memori yang berbeda pada setiap percobaan. Data menyebutkan bahwa dari segi pemakaian memori algoritma *Bubble sort* menjadi lebih sedikit seiring bertambahnya data yang diinput. Dapat dilihat pada diagram bahwa perbandingan pemakaian memori antara ketiga algoritma relatif stabil dikarenakan tidak ada perbandingan yang cukup signifikan dari ketiganya.



Gambar 2. Grafik Perbandingan Penggunaan Memori

Tabel 4. Rata-rata pemakaian memori keseluruhan percobaan

Jumlah Data	Rata-rata Pemakaian Memori (MB)		
	Bubble sort	***Selection sort***	***Introsort***
100	4,1523	4,0065	4,0143
1.000	4,1601	4,0117	4,1484
10.000	4,0456	4,0365	4,0482

Meskipun perbandingan kapasitas memori algoritma *Introsort* terlihat tidak terlalu jauh berbeda dari yang lain, jika disandingkan dengan kecepatannya, *Introsort* tetap lebih unggul sebagai algoritma paling efisien dari ketiga algoritma yang telah dikomparasikan.

D. PENUTUP

Hasil ketiga percobaan komparasi yang telah dilakukan, algoritma modern yaitu *Introsort* dapat dibuktikan sebagai algoritma pengurutan yang paling cepat daripada dua algoritma tradisional, yaitu *Bubble sort* dan *Selection sort*. *Introsort* membuktikan waktu percobaan yang lebih cepat dari ukuran dataset (100, 1.000, 10.000 data), dan membuktikan bahwa penggunaan memori *Introsort* selalu stabil dan hampir sama dengan kedua algoritma tersebut. *Bubble sort* menghasilkan

waktu percobaan yang paling lama, semakin besar dataset yang diinput maka semakin lama waktu eksekusinya. Algoritma ini kurang tepat jika digunakan untuk dataset array sedang hingga dataset array besar. *Selection sort* membuktikan waktu yang lebih cepat daripada *Bubble sort*.

Tetapi kedua algoritma tersebut, tetap tidak bisa menandingi kinerja dan kecepatan *Introsort*. Algoritma *Introsort* adalah gabungan dari *Quick Sort*, *Insertion Sort*, dan *Heap Sort* yang bisa menyesuaikan berbagai dataset sehingga memori algoritma ini selalu stabil. Dengan hasil uji percobaan ini, dapat disimpulkan bahwa algoritma *Introsort* adalah algoritma yang paling cepat digunakan untuk pengurutan data dari yang paling kecil hingga yang paling besar di dalam pemrograman C++ sesuai dengan percobaan yang dilakukan.

Untuk penelitian ke depannya disarankan untuk menggunakan dataset yang berjumlah lebih banyak atau dengan kondisi yang lebih rumit seperti data terbalik atau data dengan pola tertentu supaya bisa teranalisis lebih menyeluruh terkait performa algoritma. Disarankan untuk menguji kecepatan eksekusi dan pemakaian kapasitas memori algoritma *Introsort* lebih lanjut, seperti mengkomparasikan algoritma *Introsort* dengan algoritma modern lain seperti Timsort atau Radix Sort sehingga bisa dilihat perbandingan yang lebih komprehensif untuk aplikasi di dunia nyata. Bisa dilakukan pada penelitian berikutnya dengan mempertimbangkan penggunaan alat, misalnya dengan komputer berspesifikasi lebih tinggi. Untuk penelitian selanjutnya bisa dilakukan dengan menggunakan bahasa pemrograman lain seperti Python, Java, atau Go untuk mempertimbangkan apakah perbedaan bahasa pemrograman bisa memengaruhi kinerja algoritma. Disarankan untuk lebih mengutamakan penggunaan algoritma modern seperti *Introsort* untuk dataset berukuran besar yang membutuhkan kecepatan dan stabilitas tinggi. Jika perlu berterima kasih kepada pihak tertentu, misalnya sponsor penelitian, nyatakan dengan jelas dan singkat, hindari pernyataan terimakasih yang berlebihan.

E. 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++. *LogicLink*, *2*(1), 1–17. https://doi.org/10.28918/logiclink.v2i1.10868

Arifin, R. W., & Setiyadi, D. (2020). Algoritma Metode Pengurutan Bubble Sort dan Quick Dalam Bahasa Pemrograman C++. *Information System for Educators and Professionals*, *4*(2), 178–187. https://ejournal-binainsani.ac.id/index.php/ISBI/article/view/1348

Aulia, F., & Yahfizham. (2024). Mengenal Bahasa Pemrograman Pada Algoritma Pemrograman. *Journal Of Informatics And Busisnes*, *1*(4), 223–228. https://doi.org/10.47233/jibs.v1i1.521

Dhamma, M. (2024). Analisis Kompleksitas Diantara Algoritma Insertion Sort dan Selection Sort dan Diimplemntasikan dengan Bahasa Pemograman Java. *INFOTEKJAR : Jurnal Nasional Informatika Dan Teknologi Jaringan*, *8*(1), 6–9. https://doi.org/10.30743/infotekjar.v8i1.9641

Dharmalau, A., Rasyid, H. A., Larsen, J., Suryantoro, H., Khoiriyah, K., Nurlaela, L., Ningtyas, S., & Kurniati, I. (2025). *Buku Ajar Dasar-Dasar Teknologi Informasi: Konsep, Aplikasi, dan Perkembangannya*. Bogor : PT Mustika Sri Rosadi.

Gustina, Ahadi, A. H., Aminuddin, F. H., & Syawal, M. F. (2025). Performance Analysis of Bubble Sort, Selection Sort, and Insertion Sort Algorithms Using Python on Student Data. *JIPTI : Jurnal*

Inovasi Pendidikan Dan Teknologi Informasi, *6*(2), 529–536. https://doi.org/10.52060/jipti.v6i2.3792

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

Marcellino, Pratama, D. W., Suntiarko, S. S., & Margi, K. (2021). Comparative of Advanced Sorting Algorithms (Quick Sort, Heap Sort, Merge Sort, Intro Sort, Radix Sort) Based on Time and Memory Usage. *2021 1st International Conference on Computer Science and Artificial Intelligence (ICCSAI)*, 154–160. https://doi.org/10.1109/ICCSAI53272.2021.9609715

Murad, D. F., Sunge, A. S., Prasandy, T., Maryani, Yossy, E. H., Putranto, A., Suzanna, & Ahmad Fitriansyah. (2025). *Pemrograman Komputer: Konsep, Logika, dan Implementasi*. Bogor : PT Mustika Sri Rosadi.

Musser, D. R. (1997). Introspective sorting and selection algorithms. *Software : Practice and Experience*, *27*(8), 983–993. https://doi.org/10.1002/(SICI)1097-024X(199708)27:8%3C983::AID-SPE117%3E3.0.CO;2-%23

Nasution, R., Syahputra, A., Widiyanto, A., Subuhanto, D., & Abdillah, A. Y. (2023). Persepsi Mahasiswa Informatika Terhadap Keefektifan Algoritma Bubble Sort dalam Mengurutkan Data. *Blend Sains Jurnal Teknik*, *1*(3), 220–225. https://doi.org/10.56211/blendsains.v1i3.186

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

Rachmat, N. (2018). Perbandingan Bubble Sort, Shell Sort Dan Kombinasi Bubble Sort Dengan Shell Sort. *JUSIKOM : Jurnal Sistem Komputer Musirawas*, *3*(1), 59. https://doi.org/10.32767/jusikom.v3i1.308

Retnoningsih, E. (2018). Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort. *Information Management for Educators and Professionals*, *3*(1), 95–106. https://ejournal-binainsani.ac.id/index.php/IMBI/article/view/1060

Riki, Faridz, M., Hidayah, T. S., & Soeharsono. (2024). Perbandingan Algoritma Selection Sort, Shell Sort, dan Merge Sort Pada Data Sampling Numerik Menggunakan Matplotlib. *Prosiding Seminar Nasional Sains Dan Teknologi*, *1*, 253–265. https://conference.ut.ac.id/index.php/saintek/article/view/2616

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

Tambunan, V. F., Akva, S., Davina, S., & Indra, Z. (2024). Analisis Kinerja Insertion Sort Dan Selection Sort Dalam Pemrograman Python. *Jurnal Kajian Riset Multisiplin*, *8*(12), 33–38. https://sejurnal.com/pub/index.php/jkrm/article/view/5558

Yanti, F., & Eriana, E. S. (2024). *Algoritma Sorting Dan Searching*. Purbalingga: CV. Eureka Media Aksara.

Yovita, C., Nasution, K., & Haramaini, T. (2023). Penerapan Algoritma Naive

Bayes dan Selection Sort pada Penilaian Kuis di Aplikasi Pembelajaran Pemrograman Java dan PHP. *Hello World : Jurnal Ilmu Komputer*, *2*(3), 120–128. https://doi.org/10.56211/helloworld.v2i3.278

Yunial, A. H. (2025). Analisa Perbandingan Algoritma Bubble Sort dan Perbandingan Eksponensial. *Jurnal Informatika Universitas Pamulang*, *10*(1), 7–14.

Zahwa, S., Amelia, N. D., Nafila, R., Putri, R. A., & Pujiono, I. P. (2025). Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++. *RESTIKOM : Riset Teknik Informatika Dan Komputer*, *7*(1), 123–136. https://doi.org/10.52005/restikom.v7i1.428

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