Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

PERBANDINGAN EFISIENSI MEMORI DAN WAKTU KOMPUTASI PADA 7 ALGORITMA *SORTING* MENGGUNAKAN BAHASA PEMROGRAMAN JAVA

Imam Prayogo Pujiono[1], Rahmawan Bagus Trianto[2], Fida Maisa Hana[3]
[1]imam.prayogopujiono@uingusdur.ac.id, [2]rahmawanbagust@gmail.com, [3]fidamaisa@umkudus.ac.id
[1]Program Studi Informatika, UIN K.H. Abdurrahman Wahid Pekalongan
[2]Program Studi Ilmu Komputer, Universitas An Nuur
[3]Program Studi Sistem Informasi, Universitas Muhammadiyah Kudus

Abstrak

Perkembangan teknologi informasi telah merubah metode penyimpanan data dari fisik menjadi digital, yang menuntut pengorganisasian data yang baik untuk mempermudah pencarian dan verifikasi. Oleh karena itu, pengurutan data menjadi sangat penting dan berbagai algoritma pengurutan telah dikembangkan, seperti *Quick Sort* dan *Heap Sort*. Penelitian ini bertujuan membandingkan kinerja waktu komputasi dan penggunaan memori dari tujuh algoritma *sorting*: *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Shell Sort*, *Quick Sort*, *Merge Sort*, dan *Heap Sort* menggunakan bahasa pemrograman Java. Evaluasi dilakukan pada dataset berisi 100, 1.000, dan 10.000 data numerik acak antara 1-99. Hasil penelitian menunjukkan *Shell Sort* memberikan waktu komputasi tercepat untuk dataset berisi 100 dan 1.000 data, sementara *Heap Sort* paling efisien untuk dataset berisi 10.000 data. Dari segi penggunaan memori, ketujuh algoritma menunjukkan konsumsi memori serupa, namun *Shell Sort* membutuhkan memori lebih rendah pada dataset berisi 1.000 data, dan *Merge Sort* menggunakan memori lebih banyak pada dataset berisi 10.000 data.

Kata kunci: Algoritma *Sorting*, Efisiensi Memori, Waktu Komputasi, Java

Abstract

The development of information technology has changed data storage methods from physical to digital, which requires good data organization to make searching and verification easier. Therefore, sorting data has become very important, and various sorting algorithms have been developed, such as Quick Sort and Heap Sort. This research aims to compare the performance of computing time and memory usage of seven sorting algorithms: Bubble Sort, Insertion Sort, Selection Sort, Shell Sort, Quick Sort, Merge Sort, and Heap Sort, using the Java programming language. Evaluation is carried out on datasets containing 100, 1,000, and 10,000 random numeric data between 1-99. The research results show that Shell Sort provides the fastest computing time for datasets containing 100 and 1,000 data, while Heap Sort is the most efficient for datasets containing 10,000 data. In terms of memory usage, the seven algorithms show similar memory consumption. However, Shell Sort requires less memory on a dataset containing 1,000 data, and Merge Sort uses more memory on a dataset containing 10,000 data.

Keywords*: Sorting Algorithm, Memory Efficiency, Computing Time, Java*

1. Pendahuluan

Berkembangnya teknologi informasi telah membawa berbagai perubahan dalam kehidupan [1], seperti perubahan cara menyimpan data, mulanya lemari arsip digunakan untuk menyimpan data, tetapi sekarang data disimpan menggunakan komputer bahkan di *cloud computing* dalam bentuk data digital [2]. Seiring waktu, jumlah data yang disimpan menggunakan komputer semakin meningkat, dan data yang tidak tersusun rapi atau tidak memiliki urutan akan sulit ditemukan dan diperiksa jika terjadi kesalahan [3]. Terdapat berbagai algoritma *sorting* (pengurutan) data baik secara *ascending* (menaik) maupun *descending* (menurun) yang dapat digunakan untuk mengurutkan data [4], misalnya dengan menggunakan algoritma *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Shell Sort*, *Quick Sort*, *Merge Sort*, *Heap Sort*, dan sebagainya [3]. Algoritma *Merge Sort* dan *Quick Sort* unggul dalam mengurutkan data dalam jumlah besar, sementara algoritma *Selection Sort*, *Insertion Sort*, dan *Bubble Sort* memiliki keunggulan dalam *sorting* data dalam jumlah sedikit [5].

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

Berbagai penelitian telah dilakukan untuk menguji berbagai algoritma *sorting*. Peneliti [4] membandingkan algoritma *Insertion Sort* dan *Selection Sort* menggunakan bahasa pemrograman Java. Peneliti [5] menganalisis efektifitas waktu dan memori pada beragam algoritma *sorting* dengan bahasa pemrograman Python. Peneliti [6] mengoptimalkan algoritma *Shell Sort* ketika mengurutkan data huruf dan angka. Peneliti [7] menganalisis efektifitas waktu dan memori pada algoritma *Insertion Sort*, *Merge Sort*, dan *Heap Sort* memakai bahasa pemrograman Java. Peneliti [8] membandingkan algoritma *Bubble Sort*, *Shell Sort*, *Quick Sort* ketika mengurutkan deret angka acak memakai bahasa Java. Peneliti [9] meneliti algoritma *Bubble Sort*, *Quick Sort*, dan *Merge Sort* dalam mengurutkan kombinasi angka dan huruf. Peneliti [10] membandingkan algoritma *Quick Sort* dan *Merge Sort* dalam pengurutan data berdasarkan durasi waktu dan jumlah langkah. Peneliti [11] menganalisis algoritma *Insertion Sort* dan *Merge Sort* memakai bahasa pemrograman C++. Peneliti [12] mengevaluasi waktu komputasi algoritma *Bubble Sort* dan *Insertion Sort* memakai bahasa pemrograman Golang dan .Net C# versi 4.0.3019. Peneliti [13] membandingkan algoritma *Bubble Sort*, *Shell Sort*, dan *Quick Sort* memakai bahasa pemrograman Java. Peneliti [14] membandingkan algoritma *Insertion Sort*, *Selection Sort*, dan *Bubble Sort* menggunakan Python. Peneliti [15] mengkaji *Selection Sort* menggunakan bahasa pemrograman PHP. Peneliti [16] melakukan perbandingan antara *Quick Sort* dan *Bubble Sort* menggunakan Flutter.

Meskipun telah banyak penelitian sebelumnya, belum ditemukan penelitian yang membandingkan tujuh algoritma pengurutan secara bersamaan dengan tiga dataset berukuran berbeda. Oleh karena itu, dalam penelitian ini dilakukan pengujian terhadap algoritma *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Shell Sort*, *Quick Sort*, *Merge Sort*, dan *Heap Sort* menggunakan bahasa pemrograman Java dengan data *input* dalam jumlah yang berbeda, yaitu 100, 1.000, dan 10.000, dengan tipe data numerik acak bernilai antara 1-99. Setiap algoritma diuji berdasarkan waktu dan memori yang dibutuhkan saat mengurutkan data pada *input* data yang berbeda. Data *input* yang bervariasi ini digunakan untuk melihat perbedaan kinerja algoritma ketika mengurutkan data berukuran kecil (100), sedang (1.000), dan besar (10.000). Selanjutnya, tiap algoritma dianalisis untuk mengetahui waktu dan memori yang dibutuhkan pada setiap *input* data yang berbeda, sehingga dapat diketahui algoritma mana yang lebih efisien dalam hal kebutuhan memori dan waktu komputasi pada data berukuran kecil, sedang, dan besar.

2. Metode

Pada penelitian ini, metode yang digunakan diawali dengan pembuatan dataset sebanyak 100 data yang mewakili data kecil, 1.000 data mewakili data sedang, dan 10.000 data mewakili data besar dengan menggunakan sebuah *array*, berikut kode program yang digunakan untuk membuat sebuah *array* yang berisi dataset sebesar 100, 1.000, dan 10.000 dengan tipe data numerik acak bernilai antara 1-99.

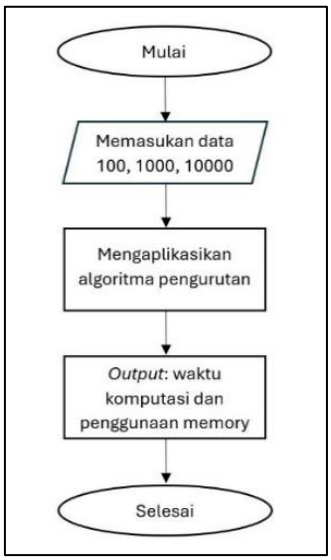
```
public static int[] createArray(10000) {
    int[] arr = new int[count];
    // Membuat array dengan ukuran 10.000
    for (int j = 0; j < count; j++) {
       // Menghasilkan angka acak dari 1 hingga 99
       arr[j] = (int) (Math.random() * 99 + 1);
    }
    // Mengembalikan array yang telah diisi dengan angka acak
    return arr;
}
```

Dari kode program di atas akan dihasilkan *array* yang berisi 10.000 elemen / 10.000 data, selanjutnya 10.000 data tersebut dimasukkan kedalam variabel bertipe *integer array* (*array of integer*) agar pengujian yang dilakukan terhadap tujuh algoritma *sorting* menggunakan data yang sama. Variabel tersebut adalah ArrayKecil, ArraySedang, ArrayBesar. ArrayKecil digunakan sebagai dataset yang berisi 100 elemen pertama dari total 10.000 elemen yang dihasilkan, ArraySedang digunakan sebagai dataset yang berisi 1.000 elemen pertama dari total 10.000 elemen yang dihasilkan, sedangkan ArrayBesar digunakan sebagai dataset yang berisi 10.000 elemen yang dihasilkan.

Kemudian, setiap algoritma *sorting* diimplementasikan dalam bahasa pemrogram Java dan diuji dengan dataset yang berisi 100 data (ArrayKecil), 1.000 data (ArraySedang), dan 10.000 data

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

(ArrayBesar). Hasil dari implementasi ini akan menghasilkan data tentang lama waktu komputasi dan besar penggunaan memori. Langkah terakhir adalah menyusun hasil / *output* dalam bentuk tabel untuk memudahkan analisis. Gambar 1 memberikan ilustrasi dari langkah-langkah dalam penelitian ini.



Gambar 1. Diagram Alir Tahapan Proses Algoritma Pengurutan

3. Hasil dan Pembahasan

Penelitian ini dilakukan memakai bahasa pemrograman Java dan IDE (*Integrated Development Environment*) NetBeans versi 14, yang berfungsi sebagai platform untuk menjalankan kode program. Spesifikasi laptop yang digunakan dalam pengujian ini adalah:

1. Windows 10 64-Bit
2. Memory: 8 GB RAM
3. Processor: Intel Core i5-9300H 2.40GHz
4. SSD berkapasitas 256 GB

Algoritma yang dipakai dalam penelitian ini yaitu algoritma *Bubble Sort*, *Insertion Sort*, *Selection Sort*, *Shell Sort*, *Quick Sort*, *Merge Sort*, dan *Heap Sort*. Semua algoritma diuji menggunakan dataset yang sama, berjumlah 100 data (ArrayKecil), 1.000 data (ArraySedang), dan 10.000 data (ArrayBesar) dengan tipe data numerik acak bernilai antara 1-99.

Dataset yang berisi 100 data digunakan untuk melakukan pengujian ke-1 hingga ke-3 pada setiap algoritma pengurutan, dataset yang berisi 1.000 data digunakan untuk pengujian ke-4 hingga ke-6, dan dataset yang berisi 10.000 data digunakan untuk pengujian ke-7 hingga ke-9 pada setiap algoritma pengurutan. Pelaksanaan pengujian sebanyak tiga kali bertujuan untuk memastikan konsistensi hasil dan meningkatkan kepercayaan terhadap hasil pengujian. Selama pengujian, hanya tiga aplikasi yang dijalankan untuk memastikan tidak ada aplikasi tambahan yang berjalan dan mempengaruhi kinerja CPU serta memori selama pengujian: NetBeans untuk menjalankan pengujian, Snipping Tools untuk mengambil *screenshot* hasil pengujian, dan Microsoft Word untuk pencatatan hasil pengujian.

Dataset pengujian dapat dilihat pada link berikut: *Link* Dataset Pengujian. Selama pengujian, setiap algoritma diukur berdasarkan lama waktu komputasi dan besar penggunaan memori. Berikut adalah link dokumentasi untuk setiap pengujian: *Link Bubble Sort*, *Link Insertion Sort*, *Link Selection Sort*, *Link Shell Sort*, *Link Quick Sort*, *Link Merge Sort*, dan *Link Heap Sort*. Berikut adalah penjelasan dari pengujian tiap algoritma.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

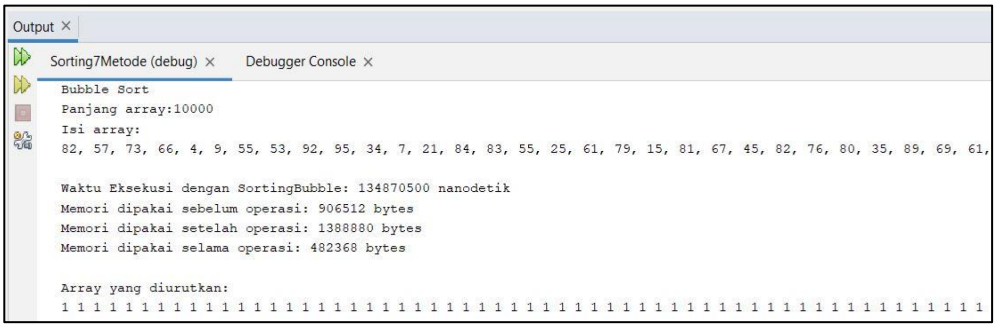
3.1. *Bubble Sort*

```
// Fungsi untuk melakukan bubble sort
        public static void bubleSort(int arr[]) {
            int p = arr.length;
            for (int j = 0; j < p-1; j++)
              for (int k = 0; k < p-j-1; k++)
                if (arr[k] > arr[k+1]) {
                  // swap arr[k+1] dan arr[k]
                  int tempo = arr[k];
                  arr[k] = arr[k+1];
                  arr[k+1] = tempo;
                }
        }
```

Kode program diatas adalah penerapan dari algoritma *Bubble Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian *Bubble Sort* dapat dilihat pada Tabel 1.

Tabel 1. Hasil Pengujian *Bubble Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	183.200	41.960
2	100	207.200	41.960
3	100	202.600	41.960
4	1.000	5.024.600	482.368
5	1.000	6.985.200	482.368
6	1.000	4.743.600	482.368
7	10.000	146.663.500	482.368
8	10.000	134.870.500	482.368
9	10.000	143.502.900	482.368



Gambar 2. Hasil Pengujian ke 8 (10.000 data) Algoritma *Bubble Sort*

Dari Tabel 1, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 197.666 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 5.584.466 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 141.678.966 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

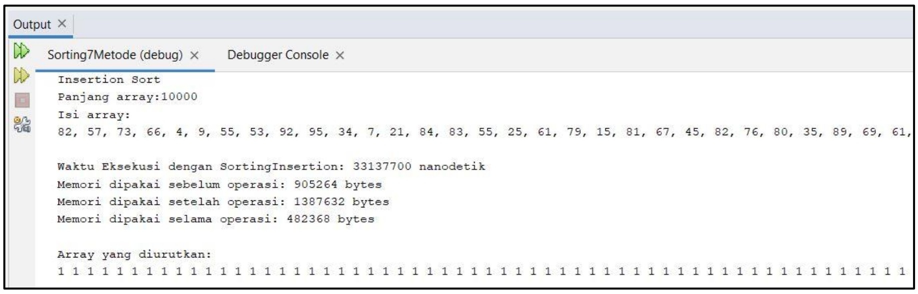
3.2. *Insertion Sort*

```
// Fungsi untuk melakukan insertion sort
        public static void insertSort(int arr[]) {
          int p = arr.length;
          for (int j = 1; j < p; j++) {
            int key = arr[j];
            int k = j - 1;

            // Memindahkan elemen yang lebih besar dari key
            // ke posisi satu langkah di depan posisi mereka saat ini
            while (k >= 0 && arr[k] > key) {
              arr[k + 1] = arr[k];
              k = k - 1;}
            arr[k + 1] = key;
          }
        }
```

Kode program di atas adalah penerapan dari algoritma *Insertion Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian *Insertion Sort* dapat dilihat pada Tabel 2.

Tabel 2. Hasil Pengujian *Insertion Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	46.900	41.960
2	100	32.800	41.960
3	100	50.100	41.960
4	1.000	5.857.200	482.368
5	1.000	4.459.600	482.368
6	1.000	4.286.100	482.368
7	10.000	33.151.100	482.368
8	10.000	33.137.700	482.368
9	10.000	32.109.900	482.368



Gambar 3. Hasil Pengujian ke 8 (10.000 data) Algoritma *Insertion Sort*

Dari Tabel 2, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 43.266 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 4.867.633 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 32.799.566 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

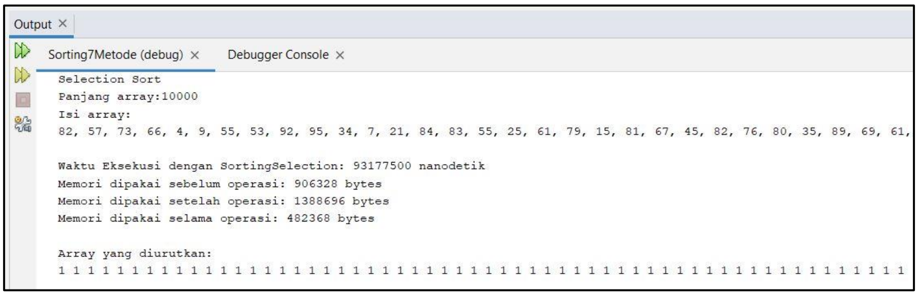
3.3. *Selection Sort*

```
// Fungsi untuk melakukan selection sort
        public static void selectSort(int arr[]) {
            int p = arr.length;

            // Satu demi satu memindahkan batas unsorted array
            for (int j = 0; j < p-1; j++) {
                // Mencari elemen minimum dalam array yang tidak tersortir
                int min_idx = j;
                for (int k = j+1; k < p; k++)
                    if (arr[k] < arr[min_idx])
                        min_idx = k;

                // Menukar elemen minimum yang ditemukan dengan elemen di posisi pertama
                int tempo = arr[min_idx];
                arr[min_idx] = arr[j];
                arr[j] = tempo;}
        }
```

Kode program di atas adalah penerapan dari algoritma *Selection Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian *Selection Sort* dapat dilihat pada Tabel 3.

Tabel 3. Hasil Pengujian *Selection Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	15.300	41.960
2	100	26.900	41.960
3	100	16.700	41.960
4	1.000	3.391.900	482.368
5	1.000	4.182.700	482.368
6	1.000	3.309.900	482.368
7	10.000	93.333.100	482.368
8	10.000	93.177.500	482.368
9	10.000	93.735.100	482.368



Gambar 4. Hasil Pengujian ke 8 (10.000 data) Algoritma *Selection Sort*

Dari Tabel 3, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 19.633 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 3.628.166 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 93.415.233 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

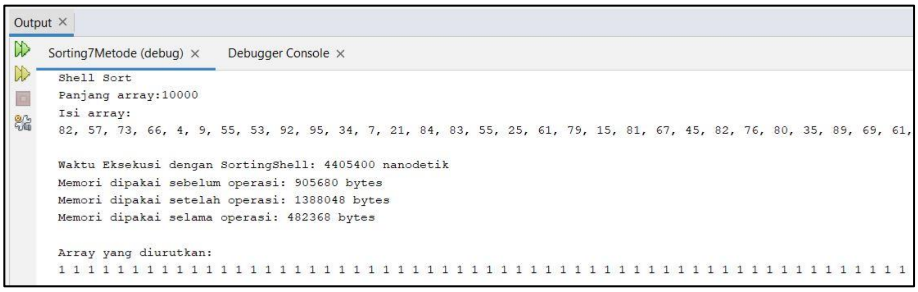
3.4. *Shell Sort*

```
// Fungsi untuk melakukan shell sort
        public static void shellSort(int arr[]) {
          int n = arr.length;

          // Mulai dengan interval yang besar, kemudian reduksi interval
          for (int gap = n/2; gap > 0; gap /= 2) {
            // Melakukan insertion sort untuk interval ini.
            for (int j = gap; j < n; j += 1) {
              int tempo = arr[j];
              int k;
              for (k = j; k >= gap && arr[k - gap] > tempo; k -= gap) {
                arr[k] = arr[k - gap];
              }
              arr[k] = tempo;
            }
          }
        }
```

Kode program di atas adalah penerapan dari algoritma *Shell Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian *Shell Sort* dapat dilihat pada Tabel 4.

Tabel 4. Hasil Pengujian *Shell Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	6.700	41.960
2	100	7.500	41.960
3	100	8.900	41.960
4	1.000	101.600	41.960
5	1.000	103.000	41.960
6	1.000	99.400	41.960
7	10.000	4.486.200	482.368
8	10.000	4.405.400	482.368
9	10.000	3.949.700	482.368



Gambar 5. Hasil Pengujian ke 8 (10.000 data) Algoritma *Shell Sort*

Dari Tabel 4, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 7.700 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 101.333 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 4.280.433 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 41.960 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

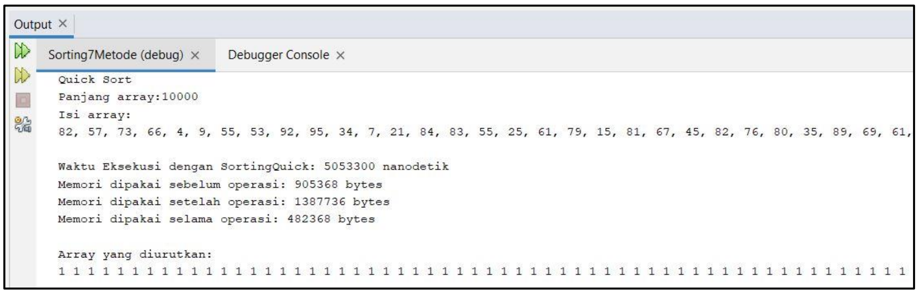
3.5. *Quick Sort*

```
// Fungsi utama yang mengimplementasikan QuickSort
        public static void quickSort(int arr[], int rendah, int tinggi) {
          if (rendah < tinggi) {
            // pi merupakan indeks partisi, arr[pi] sekarang di tempat yang tepat
            int pi = partition(arr, rendah, tinggi);

            // Secara rekursif mengurutkan elemen sebelum
            // dan setelah partisi
            quickSort(arr, rendah, pi - 1);
            quickSort(arr, pi + 1, tinggi);
          }
        }
```

Kode program di atas adalah penerapan dari algoritma *Quick Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian algoritma *Quick Sort* dapat dilihat pada Tabel 5.

Tabel 5. Hasil Pengujian *Quick Sort*

Pengujian Ke	**Jumlah Data**	**Waktu Komputasi (nanodetik)**	**Penggunaan Memori (byte)**
1	100	31.900	41.960
2	100	37.100	41.960
3	100	43.700	41.960
4	1.000	519.700	482.368
5	1.000	491.500	482.368
6	1.000	498.600	482.368
7	10.000	5.364.700	482.368
8	10.000	5.053.300	482.368
9	10.000	5.515.900	482.368



Gambar 6. Hasil Pengujian ke 8 (10.000 data) Algoritma *Quick Sort*

Dari Tabel 5, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 37.566 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 503.266 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 5.311.300 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

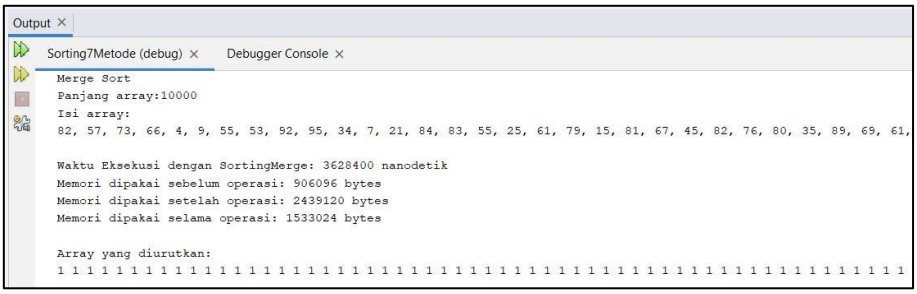
3.6. *Merge Sort*

```
// Fungsi utama yang mengimplementasikan MergeSort
public static void mergeSort(int arr[], int l, int r) {
        if (l < r) {
            // Temukan titik tengah
            int m = (l + r) / 2;

            // Urutkan setiap setengah
            mergeSort(arr, l, m);
            mergeSort(arr, m + 1, r);

            // Gabungkan kedua setengah yang telah diurutkan
            merge(arr, l, m, r);
        }
    }
```

Kode program di atas adalah penerapan dari algoritma *Merge Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian *Merge Sort* dapat dilihat pada Tabel 6.

Tabel 6. Hasil Pengujian *Merge Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	25.200	41.960
2	100	20.700	41.960
3	100	18.800	41.960
4	1.000	608.200	482.368
5	1.000	619.800	482.368
6	1.000	603.100	482.368
7	10.000	4.633.700	1.533.024
8	10.000	3.628.400	1.533.024
9	10.000	3.326.400	1.533.024



Gambar 7. Hasil Pengujian ke 8 (10.000 data) Algoritma *Merge Sort*

Dari Tabel 6, jika dilihat berdasarkan lama waktu komputasi, maka waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 21.566 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 610.366 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 3.862.833 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, maka penggunaan memori rata-rata pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 1.533.024 byte.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

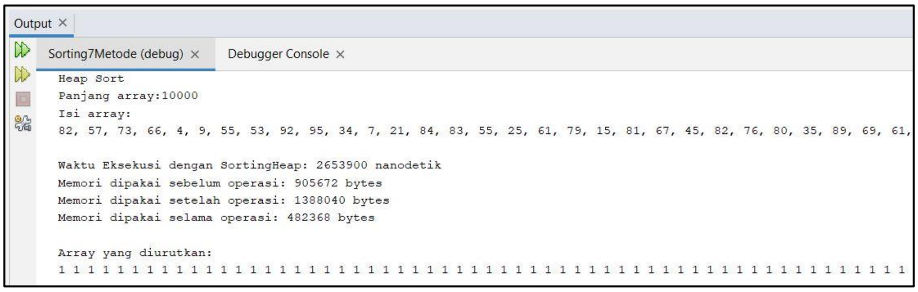
3.7. *Heap Sort*

```
// Fungsi utama untuk melakukan heap sort
        public static void heapSort(int arr[]) {
          int n = arr.length;

          // Bangun heap (atur ulang array)
          for (int j = n / 2 - 1; j >= 0; j--)
            heapify(arr, n, j);

          // Satu per satu ekstrak elemen dari heap
          for (int j=n-1; j>0; j--) {
            // Pindahkan root saat ini ke end
            int tempo = arr[0];
            arr[0] = arr[j];
            arr[j] = tempo;
            // panggil heapify pada heap yang dikurangi
            heapify(arr, j, 0);
          }
        }
```

Kode program di atas adalah penerapan dari algoritma *Heap Sort* dalam bahasa pemrograman Java. Dataset yang digunakan dimasukkan ke dalam program dan dijalankan sesuai dengan skema pengujian yang sudah ditetapkan: pengujian ke-1 sampai ke-3 menggunakan 100 data (ArrayKecil), pengujian ke-4 sampai ke-6 menggunakan 1.000 data (ArraySedang), dan pengujian ke-7 sampai ke-9 menggunakan 10.000 data (ArrayBesar). Hasil pengujian Algoritma *Heap Sort* dapat dilihat pada Tabel 7.

Tabel 7. Hasil Pengujian *Heap Sort*

Pengujian Ke	Jumlah Data	Waktu Komputasi (nanodetik)	Penggunaan Memori (byte)
1	100	16.400	41.960
2	100	19.800	41.960
3	100	18.800	41.960
4	1.000	608.400	482.368
5	1.000	607.900	482.368
6	1.000	608.100	482.368
7	10.000	2.308.400	482.368
8	10.000	2.653.900	482.368
9	10.000	2.670.500	482.368



Gambar 8. Hasil pengujian ke 8 (10.000 data) Algoritma *Heap Sort*

Dari Tabel 7, jika dilihat berdasarkan lama waktu komputasi, waktu komputasi rata-rata pada pengujian dengan jumlah data 100 adalah 18.333 nanodetik, pada pengujian dengan jumlah data 1.000 adalah 608.133 nanodetik, dan pada pengujian dengan jumlah data 10.000 adalah 2.544.266 nanodetik. Namun, jika dilihat berdasarkan besar penggunaan memori saat komputasi, penggunaan memori rata-rata

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

pada pengujian dengan jumlah data 100 adalah 41.960 byte, pada pengujian dengan jumlah data 1.000 adalah 482.368 byte, dan pada pengujian dengan jumlah data 10.000 adalah 482.368 byte.

Dari hasil pengujian tujuh algoritma yang telah dilakukan, rangkuman rata-rata hasil pengujian tiap algoritma dapat dilihat pada Tabel 8 di bawah ini.

Tabel 8. Rata-Rata Hasil Pengujian tujuh Algoritma *Sorting*

Nama Algortima	Jumlah Data	Rata-Rata Waktu Komputasi (nanodetik)	Rata-Rata Penggunaan Memori (byte)
Bubble Sort	100	197.666	41.960
Insertion Sort	100	43.266	41.960
Selection Sort	100	19.633	41.960
Shell Sort	100	7.700	41.960
Quick Sort	100	37.566	41.960
Merge Sort	100	21.566	41.960
Heap Sort	100	18.333	41.960
Bubble Sort	1.000	5.584.466	482.368
Insertion Sort	1.000	4.867.633	482.368
Selection Sort	1.000	3.628.166	482.368
Shell Sort	1.000	101.333	41.960
Quick Sort	1.000	503.266	482.368
Merge Sort	1.000	610.366	482.368
Heap Sort	1.000	608.133	482.368
Bubble Sort	10.000	141.678.966	482.368
Insertion Sort	10.000	32.799.566	482.368
Selection Sort	10.000	93.415.233	482.368
Shell Sort	10.000	4.280.433	482.368
Quick Sort	10.000	5.311.300	482.368
Merge Sort	10.000	3.862.833	1.533.024
Heap Sort	10.000	2.544.266	482.368

Berdasarkan tabel 8 yang menampilkan rata-rata hasil pengujian dari tujuh algoritma pengurutan untuk tiga kategori jumlah data yaitu 100, 1.000, dan 10.000, dapat ditarik beberapa informasi sebagai berikut:

1. Pada jumlah data 100, seluruh algoritma memperlihatkan besar penggunaan memori yang serupa, dengan nilai memori konstan sebesar 41.960 byte, dan waktu komputasi paling cepat dimiliki oleh algoritma *Shell Sort* dengan rata-rata waktu 7.700 nanodetik, sedangkan waktu komputasi paling lambat dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 197.666 nanodetik.
2. Saat jumlah data naik menjadi 1.000, Algoritma *Shell Sort* memperlihatkan penggunaan memori yang lebih sedikit dibanding algoritma lainnya yaitu 41.960 byte, waktu komputasi paling cepat juga dimiliki oleh algoritma *Shell Sort* dengan rata-rata waktu 101.333 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 5.584.466 nanodetik.
3. Saat jumlah data naik menjadi 10.000, Algoritma *Merge Sort* memperlihatkan penggunaan memori yang lebih banyak dibanding algoritma lainnya yaitu 1.533.024 byte, dan waktu komputasi paling cepat dimiliki oleh algoritma *Heap Sort* dengan rata-rata waktu 2.544.266 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 141.678.966 nanodetik.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 9, No. 2, Juli 2024
ISSN: 2715-906X (Online)
10.51717/simkom.v9i2.481

4. Kesimpulan dan Saran

4.1. Kesimpulan

Berdasarkan penelitian yang telah dilakukan, dapat disimpulkan bahwa terdapat perbedaan dalam efisiensi waktu komputasi dan penggunaan memori antara tujuh algoritma *sorting* yang diuji. *Shell Sort* merupakan algoritma dengan kinerja terbaik pada jumlah data 100 dan 1.000, hal tersebut karena pada jumlah data tersebut *Shell Sort* memiliki rata-rata waktu komputasi paling cepat dibanding algoritma lainnya dan pada jumlah data 1.000 *Shell Sort* menunjukkan penggunaan memori yang paling sedikit dibanding algoritma lainnya. Pada jumlah data 10.000, *Heap Sort* merupakan algoritma dengan rata-rata waktu komputasi paling cepat dan *Merge Sort* menunjukkan penggunaan memori yang lebih tinggi dibandingkan algoritma lain. Di sisi lain, algoritma *Bubble Sort* secara konsisten menunjukkan kinerja waktu komputasi yang paling lambat di semua ukuran dataset. Oleh karena itu, untuk melakukan *sorting* data dengan jumlah data 100 dan 1.000, algoritma terbaik yang dapat dipilih adalah *Shell Sort*, sedangkan untuk jumlah data 10.000 algoritma terbaik yang dapat dipilih adalah *Heap Sort.* Temuan ini memberikan wawasan penting untuk pemilihan algoritma *sorting* yang efisien berdasarkan efisiensi memori dan waktu komputasi pada beberapa ukuran data yang sangat berguna dalam pengembangan aplikasi berbasis Java.

4.2. Saran

Berdasarkan hasil penelitian, beberapa saran untuk penelitian mendatang adalah sebagai berikut:

1. Optimalisasi Algoritma: Mengembangkan teknik optimalisasi pada sebuah algoritma agar memiliki kinerja yang lebih efisien dari penggunaan memori dan waktu komputasi, misalnya melalui penggunaan teknik hybrid atau modifikasi algoritma.
2. Pengujian dengan Berbagai Tipe Data: Melakukan pengujian serupa dengan tipe data lain seperti string atau objek untuk melihat bagaimana algoritma berperforma dengan jenis data yang berbeda.
3. Pengujian dengan Berbagai Ukuran Data: Melakukan pengujian serupa dengan ukuran data yang lebih besar misal dengan 1.000.000 data untuk melihat bagaimana algoritma berperforma pada data yang lebih besar hal tersebut karena pada data berukuran 1.000 dan 10.000, terjadi perbedaan algoritma yang lebih efisien dari segi waktu komputasi dan penggunaan memori.

5. Ucapan Terima Kasih

Peneliti ingin menyampaikan rasa terima kasih yang mendalam kepada orang tua, keluarga, sahabat, serta rekan-rekan yang telah memberikan dukungan baik secara moril maupun materil selama proses penelitian berlangsung. Ucapan terima kasih juga ditujukan kepada para kolega yang telah memberikan masukan berharga dan kritik yang membangun selama penelitian dan penulisan. Selanjutnya, peneliti berterima kasih kepada UIN K.H. Abdurrahman Wahid Pekalongan, Universitas An Nuur, dan Universitas Muhammadiyah Kudus yang telah memberikan kesempatan untuk melaksanakan penelitian ini hingga tuntas.

Daftar Pustaka

[1] I. P. Pujiono, A. Prayogi, dan M. I. Firdausi, “Workshop Google Gemini Untuk Membuat Artikel Dengan Teknik Seo Bagi Anggota Koperasi Mahasiswa Uin K.H. Abdurrahman Wahid Pekalongan,” *Dharma Pengabdian Perguruan Tinggi (DEPATI)*, vol. 4, no. 1, hlm. 45–53, Mei 2024, doi: 10.33019/depati.v4i1.5225.

[2] S. Kurniawan, W. Wiranata, K. Kusnan, N. Ma’muriyah, dan V. V. Ting, “Pemanfaatan Komputasi Awan (Cloud Computing) Pada Bidang Pendidikan,” *Journal of Information System and Technology*, vol. 4, no. 2, hlm. 403–405, Jul 2023.

[3] S. Wijaya, F. Fauziah, dan T. W. Harjanti, “Perbandingan Algoritma Sorting dengan Menggunakan Bahasa Pemograman Javascript dalam Penggunaan Waktu Komputasi dan Penggunaan Memori,” *STRING (Satuan Tulisan Riset dan Inovasi Teknologi)*, vol. 8, no. 3, hlm. 294, Apr 2024, doi: 10.30998/string.v8i3.17972.

[4] E. Retnoningsih, “Algoritma Pengurutan Data (Sorting) Dengan Metode Insertion Sort dan Selection Sort,” *Information Management For Educators And Professionals : Journal of Information Management*, vol. 3, no. 1, hlm. 95–106, Des 2018.

[5] Y. Heryanto, F. Fauziah, dan T. W. Harjanti, "Analisis Perbandingan Ruang dan Waktu pada Algoritma Sorting Menggunakan Bahasa Pemrograman Python," *KESATRIA: Jurnal Penerapan Sistem Informasi (Komputer & Manajemen)*, vol. 4, no. 2, hlm. 342–347, Apr 2023.

[6] H. S. Tambunan, S. Sumarno, I. Gunawan, dan E. Irawan, "Optimasi Algoritma Shell Sort Dalam Pengurutan Data Huruf Dan Angka," *JUSIKOM PRIMA (Junal Sistem Informasi Ilmu Komputer Prima)*, vol. 2, no. 1, hlm. 23–27, Agu 2018.

[7] R. R. Basir, "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)*, vol. 5, no. 2, hlm. 109, Des 2020, doi: 10.30998/string.v5i2.6250.

[8] N. Sari, W. A. Gunawan, P. K. Sari, I. Zikri, dan A. Syahputra, "Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Dengan Menggunakan Bahasa Pemrograman Java," *ADI Bisnis Digital Interdisiplin Jurnal*, vol. 3, no. 1, hlm. 16–23, Jan 2022, doi: 10.34306/abdi.v3i1.625.

[9] A. Sonita dan F. Nurtaneo, "Analisis Perbandingan Algoritma Bubble Sort, Merge Sort, Dan Quick Sort Dalam Proses Pengurutan Kombinasi Angka Dan Huruf," *Pseudocode*, vol. 2, no. 2, hlm. 75–80, Agu 2016, doi: 10.33369/pseudocode.2.2.75-80.

[10] Y. Y. P. Rumapea, "Analisis Perbandingan Metode Algoritma Quick Sort Dan Merge Sort Dalam Pengurutan Data Terhadap Jumlah Langkah Dan Waktu," *METHODIKA*, vol. 3, no. 2, hlm. 5–9, Sep 2017.

[11] R. W. Arifin dan D. Setiyadi, "Algoritma Metode Pengurutan Bubble Sort dan Quick Sort Dalam Bahasa Pemrograman C++," *Information System For Educators And Professionals : Journal of Information System*, vol. 4, no. 2, hlm. 178–187, Jun 2020.

[12] M. A. Jauhari, D. Hamidin, dan M. Rahmatuloh, "Komparasi Stabilitas Eksekusi Kode Bahasa Pemrogrman .Net C# Versi 4.0.3019 Dengan Google Golang Versi 1.4.2 Menggunakan Algoritma Bubble Sort dan Insertion Sort," *Jurnal Teknik Informatika*, vol. 9, no. 1, hlm. 13–20, Jan 2017.

[13] M. L. Zulfa, M. Mikhael, dan B. N. Sari, "Analisis Perbandingan Algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam Mengurutkan Baris Angka Acak menggunakan Bahasa Java," *Jurnal Ilmiah Wahana Pendidikan*, vol. 8, no. 13, hlm. 237–246, Jun 2022.

[14] J. Iskandar, H. Suhendar, dan B. D. Pamungkas, "Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python," *G-Tech: Jurnal Teknologi Terapan*, vol. 8, no. 1, hlm. 104–113, Des 2023, doi: 10.33379/gtech.v8i1.3556.

[15] Y. A. Sandria, M. R. A. Nurhayoto, L. Ramadhani, R. S. Harefa, dan A. Syahputra, "Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP," *Hello World Jurnal Ilmu Komputer*, vol. 1, no. 4, hlm. 190–194, Des 2022, doi: 10.56211/helloworld.v1i4.187.

[16] D. R. Poetra, "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)*, vol. 9, no. 2, hlm. 806–816, Jun 2022, doi: 10.35957/jatisi.v9i2.1886.