**Jurnal Manajemen Informatika (JAMIKA)**
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

# Array Sorting Algorithm vs Algoritma Pengurutan Tradisional: Analisis Efisiensi Memori dan Waktu

## *Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis*

**Imam Prayogo Pujiono**[1*], **Eko Hari Rachmawanto**[2], **Nurul Anisa Sri Winarsih**[3]
[1]Program Studi Informatika, UIN K.H. Abdurrahman Wahid Pekalongan, Pekalongan, Indonesia
[2,3]Program Studi Teknik Informatika, Universitas Dian Nuswantoro, Semarang, Indonesia
*E-mail: imam.prayogopujiono@uingusdur.ac.id

***Abstrak***

*Perkembangan teknologi informasi telah mengubah cara kita menyimpan dan mengurutkan data. Data yang dulunya disimpan dalam lemari arsip kini tersimpan dalam bentuk digital. Namun, data digital yang tidak teratur dapat menyulitkan proses pencarian dan verifikasi. Oleh karena itu, pengurutan data menjadi penting dan berbagai algoritma pengurutan telah dikembangkan untuk memenuhi kebutuhan ini, seperti algoritma Array Sorting Algorithm (ASA) yang diklaim memiliki kompleksitas waktunya yang efisien dan unggul jika dibandingkan dengan algoritma tradisional. Penelitian ini mengkaji efisiensi memori dan waktu komputasi antara ASA dengan lima algoritma pengurutan tradisional menggunakan bahasa pemrograman Java. Penelitian dilakukan dengan dataset numerik acak dalam tiga ukuran berbeda (100, 1.000, dan 10.000 data). ASA, dengan pendekatannya menggunakan array dua dimensi, menunjukkan performa yang mengesankan dalam hal waktu komputasi, khususnya pada dataset 1.000 dan 10.000 data, dengan rata-rata waktu komputasi masing-masing adalah 67.033 nanodetik dan 572.300 nanodetik, namun kalah pada bagian pemakaian memori. Oleh karena itu, jika penggunaan memori tidak menjadi pertimbangan maka ASA adalah algoritma pengurutan yang cocok untuk data berjumlah 1.000 – 10.000 karena memiliki rata-rata waktu komputasi lebih cepat dibanding lima algoritma pengurutan tradisional. Temuan ini memberikan wawasan penting untuk memilih algoritma pengurutan berdasarkan efisiensi memori dan waktu komputasi dalam mengembangkan aplikasi berbasis Java.*

***Kata kunci:*** *Algoritma Pengurutan, Array Sorting Algorithm, Efisiensi Memori, Java, Waktu Komputasi.*

***Abstract***

*The development of information technology has changed the way we store and sort data. Data that used to be stored in filing cabinets is now stored in digital form. However, disorganised digital data can make it difficult to search and verify. Therefore, data sorting is important, and various sorting algorithms have been developed to fulfil this need, such as the Array Sorting Algorithm (ASA), which is claimed to have efficient and superior time complexity compared to traditional algorithms. This study examines the memory efficiency and computation time between ASA and five traditional sorting algorithms using Java programming. The study was conducted with random numerical datasets in three different sizes (100, 1,000, and 10,000 data). ASA, with its approach of using two-dimensional arrays, showed impressive performance in terms of computation time, especially on the 1,000 and 10,000 datasets, with average computation times of 67,033 nanoseconds and 572,300 nanoseconds, respectively, but lost out on the memory usage part. Therefore, if memory usage is not a consideration then ASA is a suitable sorting algorithm for 1,000 - 10,000 data as it has a faster average computation time than the five traditional sorting algorithms. These findings provide insights for selecting sorting algorithms based on memory efficiency and computation time in developing Java-based applications.*

***Keywords:*** *Sorting Algorithm, Array Sorting Algorithm, Memory Efficiency, Java, Computation Time.*

*Naskah diterima 25 Jun. 2024; direvisi 26 Jul. 2024; dipublikasikan 01 Apr. 2025.*

Jurnal JAMIKA Manajemen Informatika

**Jurnal Manajemen Informatika (JAMIKA)**
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

## I. PENDAHULUAN

Perkembangan teknologi informasi telah mendatangkan perubahan signifikan dalam berbagai aspek kehidupan [1][2], termasuk dalam cara kita menyimpan data. Dulu data disimpan dalam lemari arsip, namun sekarang telah beralih ke penyimpanan digital menggunakan komputer dan *cloud computing* [3]. Seiring peningkatan volume data yang disimpan secara digital, memiliki data yang terorganisir (terurut) dengan baik menjadi semakin krusial. Data yang tidak terurut akan menyulitkan proses pencarian dan verifikasi [4]. Untuk mengatasi hal ini berbagai algoritma pengurutan data telah dikembangkan baik untuk mengurutkan data dalam urutan menaik maupun menurun [5]. Beberapa algoritma tersebut adalah *Heap Sort*, *Quick Sort*, *Shell Sort*, *Merge Sort*, *Selection Sort*, *Insertion Sort*, dan *Bubble Sort* [6]. Algoritma seperti *Quick Sort* dan *Merge Sort* sangat efektif untuk mengurutkan volume data yang besar, sementara *Selection Sort*, *Insertion Sort*, dan *Bubble Sort* lebih cocok untuk jumlah data yang lebih kecil [7].

Setiap algoritma pengurutan di ilmu komputer dirancang berdasarkan beberapa strategi utama, termasuk metode perbandingan versus non-perbandingan, pendekatan iteratif versus rekursif, serta teknik membagi dan menaklukkan [8]. Algoritma pengurutan yang menggunakan perbandingan meliputi *Quick Sort*, *Merge Sort*, *Insertion Sort*, *Selection Sort*, dan *Bubble Sort* [9]. Sementara itu, algoritma pengurutan yang tidak menggunakan perbandingan diantaranya *Counting Sort* dan *Radix Sort* [10].

Efisiensi memori dan waktu komputasi adalah faktor penting dalam penilaian algoritma pengurutan [11]. Memilih algoritma yang tepat tidak hanya dapat mengurangi penggunaan memori tetapi juga mempercepat waktu proses [12], oleh karena itu Sangat penting untuk menguasai berbagai algoritma pengurutan [13]. Banyak algoritma pengurutan baru yang berfokus pada efisiensi memori dan waktu komputasi, salah satunya adalah *Array Sorting Algorithm* (ASA).

*Array Sorting Algorithm* (ASA) adalah inovasi terbaru dalam algortima pengurutan yang dirancang khusus untuk mengurutkan bilangan bulat positif. ASA menggunakan struktur array dua dimensi, dimana kolom pertama menyimpan nilai elemen yang akan diurutkan, sedangkan kolom kedua berfungsi untuk menghitung frekuensi elemen. Pendekatan ini memungkinkan ASA untuk mengelola dan mengurutkan elemen yang sering muncul secara efisien, dengan mencatat frekuensi kemunculan mereka. Keunggulan utama ASA terletak pada kompleksitas waktunya yang efisien, yaitu antara O(n) pada kasus terbaik dan O(2n) pada kasus terburuk, yang sangat kompetitif jika dibandingkan dengan kompleksitas O($n$ log $n$) yang umum pada algoritma pengurutan tradisional [8]. ASA hanya bekerja pada bilangan bulat positif, yang umumnya mencakup data penting dalam berbagai hal di dunia nyata seperti nomor KTP, telepon, rekening bank, dan NIM mahasiswa. Berikut ini adalah *Pseudocode* dari algoritma ASA [8]:

1. *Find the largest element in the array*
2. *Declare a two-dimensional array [Large+1,2]*
3. *Set all elements of the two-dimensional array to be zeroes*
4. *Counter=0*
5. *Input the elements to be sorted*
6. *For i inside [Large+1]*
   *Array [element, counter++] = element*
7. *Remove element with counter=0*
8. *Print array*

*N.B: in case of non-repeated elements, one was declared one-dimensional array without counter.*

Banyak penelitian yang telah dilakukan untuk menentukan algoritma mana yang paling efisien pada sebuah konsidi. Sebuah penelitian [11] menganalisis kompleksitas ruang dan waktu dari algoritma *Insertion Sort*, *Merge Sort*, dan *Heap Sort* menggunakan bahasa pemrograman Java. Penelitian lain [14] meneliti kompleksitas waktu algoritma pengurutan yang digunakan untuk mengurutkan harga dari yang terendah hingga tertinggi di *platform* Shopee. Penelitian yang dilakukan pada tahun 2022 [15] membandingkan performa algoritma *Bubble Sort*, *Shell Sort*, dan *Quick Sort* dalam mengurutkan deret angka acak dengan menggunakan Java. Penelitian lainnya [16] membandingkan algoritma pengurutan sederhana dengan dan tanpa penggunaan variabel sementara. Peneliti lain yang dilakukan pada tahun 2020 [17] menganalisis algoritma *Insertion Sort* dan *Merge Sort* memakai bahasa pemrograman C++. Penelitian lainnya [18] membandingkan kecepatan antara *Selection Sort* dan *Bubble Sort* pada data yang ukurannya berkisar antara 100 hingga 100.000. Adapun penelitian pada tahun 2022 [19] membandingkan algoritma *Shell Sort*, *Bubble Sort*, dan *Quick Sort* untuk mengurutkan baris angka acak memakai bahasa pemrograman Java. Penelitian pada tahun 2022 [20] membandingkan performa *Quick Sort* dan *Bubble Sort* pada bahasa pemrograman Flutter dan Dart SDK.

Jurnal JAMIKA Manajemen Informatika
**Jurnal Manajemen Informatika (JAMIKA)**
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

Penelitian lain yang berkenaan dengan *sorting* [21] mengkaji *Selection Sort* memakai bahasa pemrograman PHP. Penelitian yang juga mengkaji tentang *sorting* [22] membandingkan algoritma *Insertion Sort*, *Selection Sort*, dan *Bubble Sort* dengan memakai Python sebagai bahasa pemrogramannya. Selain itu, penelitian lainnya [23] membandingkan efisiensi memori dan waktu komputasi pada tujuh algoritma pengurutan tradisional menggunakan bahasa pemrogramnan java.

Pada penelitian ini, dilakukan perbandingan efisiensi memori dan waktu komputasi antara *Array Sorting Algorithm* (ASA) dengan lima algoritma pengurutan tradisional, yaitu *Bubble Sort*, *Shell Sort*, *Merge Sort*, *Quick Sort*, dan *Heap Sort* menggunakan bahasa pemrograman Java. Data input menggunakan tipe data numerik acak dengan nilai antara 1-99 dalam tiga ukuran berbeda: 100, 1.000, dan 10.000. Efisiensi memori diukur berdasarkan jumlah memori yang dibutuhkan komputer untuk menjalankan sebuah algoritma, sedangkan waktu komputasi diukur dari durasi yang dibutuhkan komputer untuk menyelesaikan proses pengurutan. Data input yang bervariasi digunakan untuk menguji kinerja keenam algoritma dalam mengurutkan data berukuran kecil (100), sedang (1.000), dan besar (10.000). Selanjutnya, kinerja ASA akan dianalisis dan dibandingkan dengan kelima algoritma lainnya untuk menentukan apakah ASA lebih efisien dalam hal pemakaian memori dan waktu komputasi pada pemgurutan data berjumlah 100, 1.000, dan 10.000.

## II. METODE PENELITIAN

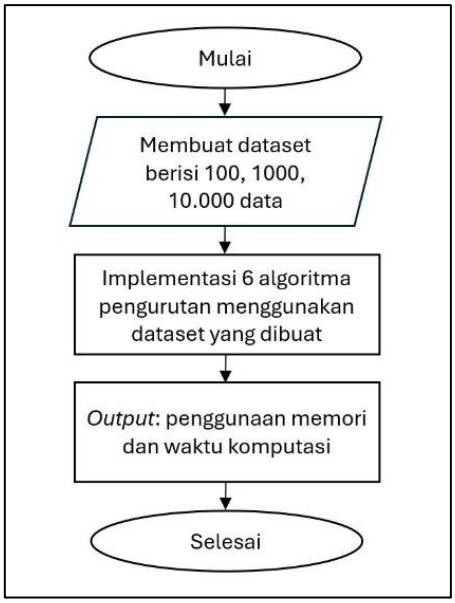
Dalam penelitian ini, metode yang digunakan diawali dengan membuat dataset dalam tiga ukuran berbeda menggunakan array, yaitu dataset sebanyak 100 data untuk ukuran kecil, 1.000 data untuk ukuran sedang, dan 10.000 data untuk ukuran besar. Setiap dataset berisi angka numerik acak yang nilainya berkisar antara 1 hingga 99. Kode program yang digunakan untuk membuat sebuah array yang nantinya digunakan untuk membuat dataset sebesar 100, 1.000, dan 10.000 dengan tipe data numerik acak bernilai antara 1-99 dapat dilihat dibawah ini:

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

Dari kode program diatas, akan dihasilkan array yang berisi 10.000 elemen. Kemudian, elemen-elemen ini dimasukkan ke dalam tiga variabel bertipe integer array (*array of integer*), yang digunakan untuk melakukan pengujian terhadap algortima ASA, *Bubble Sort*, *Shell Sort*, *Merge Sort*, *Quick Sort*, dan *Heap Sort* menggunakan data yang sama. Ketiga variabel tersebut adalah ArraySmall, ArrayMedium, dan ArrayLarge. ArraySmall berisi 100 elemen pertama dari total 10.000 elemen, ArrayMedium berisi 1.000 elemen pertama, dan ArrayLarge berisi seluruh 10.000 elemen.

Setiap algoritma pengurutan kemudian diimplementasikan dalam bahasa pemrograman Java dan diuji menggunakan ketiga dataset untuk mengukur lama waktu komputasi dan besar pemakaian memori dari setiap algoritma. Hasil pengujian disusun dalam tabel dan grafik untuk memudahkan analisis komparatif antar algoritma berdasarkan efisiensi waktu dan pemakaian memori. Gambar 1 menunjukkan gambaran dari tahapan pada penelitian ini.

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika




Gambar 1. Diagram Alur Tahapan Penelitian

## III. HASIL DAN PEMBAHASAN

Penelitian ini dilakukan memakai bahasa pemrograman Java dan *Integrated Development Environment* (IDE) NetBeans versi 14 untuk mengembangkan serta menjalankan kode program. Pengujian dijalankan pada laptop dengan spesifikasi sebagai berikut:

- Prosesor Intel Core i5-9300H dengan kecepatan 2.40GHz
- Memori 8 GB RAM
- Sistem operasi Windows 10 64-Bit
- SSD berkapasitas 256 GB

Algoritma yang diuji dalam penelitian ini meliputi ASA, *Bubble Sort*, *Shell Sort*, *Merge Sort*, *Quick Sort*, dan *Heap Sort*. Ke enam algoritma diuji dengan dataset yang sama, terdiri dari 100 data (ArraySmall), 1.000 data (ArrayMedium), dan 10.000 data (ArrayLarge) dengan tipe data numerik acak bernilai antara 1-99. Setiap algoritma menjalani tiga kali pengujian:

- Pengujian pertama hingga ketiga menggunakan dataset 100 data (ArraySmall)
- Pengujian keempat hingga keenam menggunakan dataset 1.000 data (ArrayMedium)
- Pengujian ketujuh hingga kesembilan menggunakan dataset 10.000 data (ArrayLarge)

Pengujian dilakukan tiga kali pada setiap dataset bertujuan untuk menjamin konsistensi hasil dan memperkuat keyakinan terhadap data yang diperoleh dari pengujian. Selama proses pengujian, hanya tiga aplikasi yang dijalankan untuk memastikan tidak terjadi gangguan dari aplikasi lain yang bisa mempengaruhi kinerja CPU dan memori. Ketiga aplikasi tersebut adalah NetBeans yang digunakan untuk melaksanakan pengujian, Snipping Tools yang berfungsi untuk mengambil tangkapan layar hasil pengujian, dan Microsoft Word yang dipakai untuk mendokumentasikan hasil pengujian.

Dataset pengujian dapat dilihat pada link berikut: bit.ly/4c2Fzul. Selama pengujian, setiap algoritma diukur berdasarkan lama waktu komputasi dan besar pemakaian memori. Berikut adalah link dokumentasi untuk setiap pengujian: Link ASA (bit.ly/3SiOctx), Link *Bubble Sort* (bit.ly/3M06sER), Link *Shell Sort* (bit.ly/4fh8fmd), Link *Merge Sort* (bit.ly/4djKb0m), Link *Quick Sort* (bit.ly/4dly3ff), dan Link *Heap Sort* (bit.ly/3SnCv52). Selanjutnya, hasil pengujian tiap algoritma akan dijelaskan lebih rinci.

***Array Sorting Algorithm* (ASA)**

```
// Fungsi utama ASA dalam bahasa pemrograman Java
public static void SortingASA(int[] arr) {
    // Menemukan nilai maksimal dalam array
    // untuk menentukan ukuran array dua dimensi
    int max = findMax(arr);
```

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

```
    // Array dua dimensi untuk menyimpan frekuensi elemen
    // Kolom pertama untuk elemen & kolom kedua untuk menghitung frekuensi
    int[][] freqArray = new int[max + 1][2];

    // Mengisi array dengan elemen dan frekuensinya
    for (int element : arr) {
       freqArray[element][0] = element; // Menyimpan elemen
       freqArray[element][1] += 1;      // Menghitung frekuensi
    }

    // Mencetak elemen-elemen dalam urutan terurut
    int index = 0;
    for (int i = 0; i < freqArray.length; i++) {
       while (freqArray[i][1] > 0) {
          arr[index++] = i;
          freqArray[i][1]--; // Mengurangi penghitung setelah mencetak
       }
    }
}
```

Kode program di atas merupakan implementasi dari algoritma ASA menggunakan bahasa pemrograman Java. Algoritma ini menggunakan prinsip yang mempunyai kemiripan dengan *Counting Sort* untuk mengurutkan array. Inti dari algoritma ASA adalah menciptakan sebuah array dua dimensi. freqArray pada algoritma ini berfungsi untuk menampung dan menghitung frekuensi masing-masing elemen dalam array asli. Langkah pertama dalam algoritma adalah menemukan nilai maksimal dalam array, yang digunakan untuk menentukan ukuran freqArray. freqArray memiliki dua kolom; kolom pertama digunakan untuk menyimpan elemen itu sendiri, sementara kolom kedua untuk menyimpan berapa kali elemen tersebut muncul dalam array.

Setelah menentukan nilai maksimal dan mempersiapkan freqArray, proses selanjutnya adalah mengisi freqArray dengan mengulang setiap elemen dalam array asli dan mencatat frekuensinya. Setiap kali elemen ditemukan, nilai frekuensi di freqArray untuk elemen tersebut ditingkatkan. Setelah semua elemen dihitung, algoritma kemudian mengulang dari elemen terkecil hingga terbesar dalam freqArray untuk "mencetak ulang" elemen-elemen ini kembali ke dalam array asli berdasarkan jumlah frekuensinya. Hal ini dilakukan dengan mengurangi penghitung frekuensi setiap kali elemen dicetak, memastikan setiap elemen diposisikan pada urutan yang tepat dalam array. Proses ini menghasilkan array yang terurut dari nilai terkecil ke terbesar. Pendekatan ini sangat efisien untuk dataset dengan rentang nilai yang kecil atau untuk dataset yang elemennya memiliki banyak nilai yang sama karena dapat meminimalisir perbandingan langsung antar elemen.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma ASA dapat dilihat pada Tabel 1, sementara dokumentasi pengujian dapat dilihat di: bit.ly/3SiOctx.

TABEL 1
HASIL PENGUJIAN ALGORITMA ASA

| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 41.960 | 19.800 |
| 2 | 100 | 41.960 | 14.000 |
| 3 | 100 | 41.960 | 12.700 |
| 4 | 1.000 | 482.368 | 76.300 |
| 5 | 1.000 | 482.368 | 62.800 |
| 6 | 1.000 | 482.368 | 62.000 |
| 7 | 10.000 | 964.736 | 635.700 |
| 8 | 10.000 | 964.736 | 535.300 |
| 9 | 10.000 | 964.736 | 545.900 |

Berdasarkan Tabel 1, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 41.960 byte, dengan 1.000 data adalah 482.368 byte, dan dengan 10.000

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

data adalah 964.736 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 15.500 nanodetik, dengan 1.000 data adalah 67.033 nanodetik, dan dengan 10.000 data adalah 572.300 nanodetik.

***Bubble Sort***

```
// Fungsi utama Bubble Sort dalam bahasa pemrograman Java
public static void SortingBubble(int arr[]) {
    int p = arr.length;
    for (int a = 0; a < p-1; a++)
       for (int b = 0; b < p-a-1; b++)
          if (arr[b] > arr[b+1]) {
             // swap arr[b+1] dan arr[b]
             int temp = arr[b];
             arr[b] = arr[b+1];
             arr[b+1] = temp;
          }
}
```

Kode program di atas merupakan implementasi dari algoritma *Bubble Sort* dalam bahasa pemrograman Java. Algoritma ini bekerja dengan membandingkan setiap pasangan elemen yang bersebelahan dalam array dan menukarnya jika urutannya tidak benar. Proses ini diulang untuk setiap elemen kecuali yang terakhir dalam iterasi pertama, kemudian untuk setiap elemen kecuali dua terakhir dalam iterasi kedua, dan seterusnya, secara efektif "mengapungkan" elemen terbesar ke bagian akhir array pada setiap putaran. Pengulangan ini terus berlangsung hingga tidak ada lagi pertukaran yang dibutuhkan, yang menandakan bahwa array telah sepenuhnya terurut. Walaupun *Bubble Sort* merupakan algoritma yang sederhana dan mudah dipahami, namun algoritma ini kurang efisien untuk data skala besar karena memiliki kompleksitas waktu kuadratik $O(n^2)$. Ini menjadikannya kurang praktis dibandingkan dengan algoritma *sorting* yang lebih canggih seperti *Quick Sort* atau *Merge Sort*, terutama pada data yang jumlahnya banyak atau data yang sudah hampir terurut, dimana algoritma lain biasanya jauh lebih cepat.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma *Bubble Sort* dapat dilihat pada Tabel 2, sementara dokumentasi pengujian dapat dilihat di: bit.ly/3M06sER.

TABEL 2
HASIL PENGUJIAN ALGORITMA *BUBBLE SORT*

| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 41.960 | 156.500 |
| 2 | 100 | 41.960 | 180.800 |
| 3 | 100 | 41.960 | 192.100 |
| 4 | 1.000 | 482.368 | 4.981.700 |
| 5 | 1.000 | 482.368 | 5.165.900 |
| 6 | 1.000 | 482.368 | 4.924.700 |
| 7 | 10.000 | 482.368 | 125.519.200 |
| 8 | 10.000 | 482.368 | 123.379.600 |
| 9 | 10.000 | 482.368 | 123.704.000 |

Berdasarkan Tabel 2, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 41.960 byte, dengan 1.000 data adalah 482.368 byte, dan dengan 10.000 data adalah 482.368 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 176.466 nanodetik, dengan 1.000 data adalah 5.024.100 nanodetik, dan dengan 10.000 data adalah 124.200.933 nanodetik.

***Shell Sort***

```
// Fungsi utama Shell Sort dalam bahasa pemrograman Java
public static void SortingShell(int arr[]) {
```

```
    int p = arr.length;

    // Mulai dari interval yang besar, selanjutnya kurangi interval
    for (int gap = p/2; gap > 0; gap /= 2) {
      // Melakukan insertion sort untuk interval ini.
      for (int a = gap; a < p; a += 1) {
        int temp = arr[a];
        int b;
        for (b = a; b >= gap && arr[b - gap] > temp; b -= gap) {
          arr[b] = arr[b - gap];
        }
        arr[b] = temp;
      }
    }
}
```

Kode program di atas merupakan implementasi dari algoritma *Shell Sort* menggunakan bahasa pemrograman Java. Algoritma ini merupakan peningkatan dari algoritma *Insertion Sort* dengan mengenalkan konsep "gap". Awalnya, *Shell Sort* menetapkan sebuah gap besar, yang merupakan setengah dari panjang array. Elemen-elemen dalam array kemudian dibandingkan dan diurutkan berdasarkan gap ini, memungkinkan perbandingan dan pertukaran elemen yang jauh terpisah. Setelah satu siklus dengan gap tertentu, gap tersebut berkurang (umumnya dipecah menjadi dua), dan proses serupa diulang hingga gap menjadi nol. Proses ini efektif mempercepat pengurutan dengan meminimalisir jumlah pertukaran yang diperlukan pada tahapan akhir pengurutan ketika gap menjadi sangat kecil, menjadikan *Shell Sort* lebih efisien dibandingkan dengan *Insertion Sort*, terutama untuk dataset besar.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma *Shell Sort* dapat dilihat pada Tabel 3, sementara dokumentasi pengujian dapat dilihat di: bit.ly/4fh8fmd.

TABEL 3
HASIL PENGUJIAN ALGORITMA *SHELL SORT*

| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 41.960 | 25.100 |
| 2 | 100 | 41.960 | 32.400 |
| 3 | 100 | 41.960 | 27.300 |
| 4 | 1.000 | 41.960 | 82.100 |
| 5 | 1.000 | 41.960 | 79.700 |
| 6 | 1.000 | 41.960 | 79.200 |
| 7 | 10.000 | 482.368 | 3.869.700 |
| 8 | 10.000 | 482.368 | 4.091.700 |
| 9 | 10.000 | 482.368 | 4.025.900 |

Berdasarkan Tabel 3, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 41.960 byte, dengan 1.000 data adalah 41.960 byte, dan dengan 10.000 data adalah 482.368 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 28.266 nanodetik, dengan 1.000 data adalah 80.333 nanodetik, dan dengan 10.000 data adalah 3.995.766 nanodetik.

***Merge Sort***

```
// Fungsi utama Merge Sort dalam bahasa pemrograman Java
public static void SortingMerge(int arr[], int l, int r) {
    if (l < r) {
      // Temukan titik tengah
      int m = (l + r) / 2;
```

```
        // Mengurutkan setiap setengah
        SortingMerge(arr, l, m);
        SortingMerge(arr, m + 1, r);

        // Menggabungkan kedua bagian yang sudah diurutkan
        merge(arr, l, m, r);
    }
}
```

Kode program di atas merupakan implementasi dari algoritma *Merge Sort* menggunakan bahasa pemrograman Java. Algortima ini memakai pendekatan *divide and conquer*. Prosesnya dimulai dengan memecah array menjadi dua bagian yang lebih kecil secara rekursif sampai setiap bagian hanya terdiri dari satu elemen atau tidak ada elemen sama sekali. Setelah itu, dua bagian yang sudah terurut ini akan digabungkan kembali menjadi satu array dengan cara mengurutkan elemen-elemen dari kedua bagian tersebut. Proses penggabungan ini dijalankan dengan membandingkan elemen-elemen dari kedua bagian tersebut dan menempatkan elemen yang lebih kecil terlebih dahulu dalam array yang baru. Langkah ini diulang sampai seluruh array digabungkan kembali menjadi satu dan dalam keadaan terurut. Algoritma ini memiliki kecepatan $O(n \log n)$, membuatnya efisien untuk pengurutan data dalam jumlah besar.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma *Merge Sort* dapat dilihat pada Tabel 4, sementara dokumentasi pengujian dapat dilihat di: bit.ly/4djKb0m.

TABEL 4
HASIL PENGUJIAN ALGORITMA *MERGE SORT*

| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 482.368 | 51.300 |
| 2 | 100 | 482.368 | 52.200 |
| 3 | 100 | 482.368 | 51.600 |
| 4 | 1.000 | 482.368 | 655.200 |
| 5 | 1.000 | 482.368 | 596.500 |
| 6 | 1.000 | 482.368 | 620.900 |
| 7 | 10.000 | 965.976 | 3.523.500 |
| 8 | 10.000 | 965.976 | 3.882.600 |
| 9 | 10.000 | 965.976 | 3.286.000 |

Berdasarkan Tabel 4, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 482.368 byte, dengan 1.000 data adalah 482.368 byte, dan dengan 10.000 data adalah 965.976 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 51.700 nanodetik, dengan 1.000 data adalah 624.200 nanodetik, dan dengan 10.000 data adalah 3.564.033 nanodetik.

***Quick Sort***

```
// Fungsi utama Quick Sort dalam bahasa pemrograman Java
public static void SortingQuick(int arr[], int low, int high) {
    if (low < high) {
        // pi merupakan indeks partisi, dan arr[pi] saat ini berada pada posisi yang benar
        int pi = partition(arr, low, high);
        // Secara rekursif mengurutkan elemen sebelum dan sesudah partisi
        SortingQuick(arr, low, pi - 1);
        SortingQuick(arr, pi + 1, high);
    }
}
```

Kode program di atas merupakan implementasi dari algoritma *Quick Sort* menggunakan bahasa pemrograman Java. Algoritma ini bekerja menggunakan prinsip *divide and conquer* untuk mengurutkan array. Ini dimulai dengan memilih elemen pivot melalui fungsi partition, yang mengatur ulang array sehingga elemen-

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

elemen yang lebih besar dari pivot berada di kanan pivot, dan yang lebih kecil di kiri. Setelah pivot ditempatkan pada posisi yang benar, fungsi *Quick Sort* dipanggil secara rekursif untuk mengurutkan subarray di kiri dan kanan pivot. Proses ini diulangi hingga semua elemen berada pada posisi yang benar. Algoritma ini sangat efisien dengan kompleksitas waktu rata-rata $O(n \log n)$, namun efisiensinya bergantung pada posisi pivot, dan pada kasus terburuk, kompleksitasnya dapat meningkat menjadi $O(n^2)$. Untuk menghindari kasus terburuk ini, implementasi modern *Quick Sort* sering kali menggunakan teknik seperti memilih pivot secara acak, atau menggunakan "median-of-three" untuk memilih pivot.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma *Quick Sort* dapat dilihat pada Tabel 5, sementara dokumentasi pengujian dapat dilihat di: bit.ly/4dly3ff.

TABEL 5
HASIL PENGUJIAN ALGORITMA *QUICK SORT*

| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 41.960 | 24.900 |
| 2 | 100 | 41.960 | 26.200 |
| 3 | 100 | 41.960 | 25.800 |
| 4 | 1.000 | 482.368 | 554.100 |
| 5 | 1.000 | 482.368 | 567.500 |
| 6 | 1.000 | 482.368 | 547.700 |
| 7 | 10.000 | 482.368 | 5.370.300 |
| 8 | 10.000 | 482.368 | 5.352.300 |
| 9 | 10.000 | 482.368 | 5.264.100 |

Berdasarkan Tabel 5, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 41.960 byte, dengan 1.000 data adalah 482.368 byte, dan dengan 10.000 data adalah 482.368 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 25.633 nanodetik, dengan 1.000 data adalah 556.433 nanodetik, dan dengan 10.000 data adalah 5.328.900 nanodetik.

***Heap Sort***

```
// Fungsi utama Heap Sort dalam bahasa pemrograman Java
public static void SortingHeap(int arr[]) {
    int p = arr.length;

    // Bangun heap (mengatur ulang array)
    for (int a = p / 2 - 1; a >= 0; a--)
       heapify(arr, p, a);

    // Ekstrak elemen dari heap satu per satu
    for (int a=p-1; a>0; a--) {
       // Pindahkan root saat ini ke end
       int temp = arr[0];
       arr[0] = arr[a];
       arr[a] = temp;

       // Memanggil fungsi heapify pada heap yang telah dikurangi
       heapify(arr, a, 0);
    }
}
```

Kode program di atas merupakan implementasi dari algoritma *Heap Sort* menggunakan bahasa pemrograman Java. Algortima ini berjalan dengan mengatur ulang array menjadi struktur heap menggunakan proses yang disebut "heapify", dimulai dari tengah array ke arah depan untuk memastikan bahwa setiap elemen induk lebih besar dari anak-anaknya, menciptakan max heap. Setelah heap terbentuk, elemen paling atas heap

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika

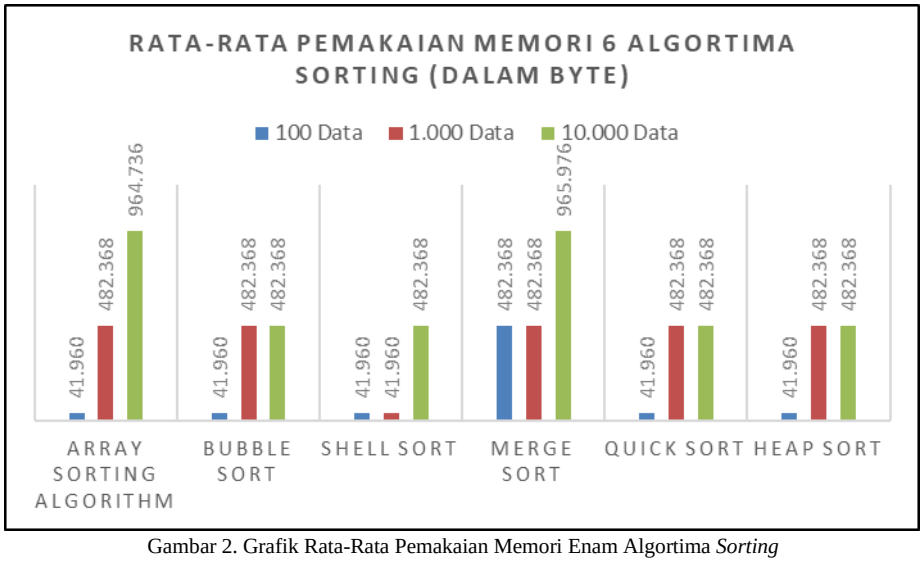
(yang terbesar) dipindahkan ke akhir array yang belum urut, dan proses heapify diulang untuk sisa array yang belum terurut, sehingga memindahkan elemen terbesar berikutnya ke tempat yang benar. Proses ini diulangi hingga semua elemen telah diurutkan, dengan heapify dipanggil berulang kali untuk mempertahankan struktur heap setelah setiap penghapusan elemen, yang memastikan bahwa array diurutkan secara efektif dari elemen terbesar ke terkecil yang kemudian menghasilkan urutan akhir dari terkecil ke terbesar. Ini adalah metode yang efisien dengan kompleksitas waktu $O(n \log n)$, menjadikannya cocok untuk mengurutkan data dalam jumlah besar.

Dataset yang digunakan nantinya akan dimasukkan ke dalam program sesuai dengan skenario pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. Hasil dari pengujian algoritma *Heap Sort* dapat dilihat pada Tabel 6, sementara dokumentasi pengujian dapat dilihat di: bit.ly/3SnCv52.

TABEL 6
HASIL PENGUJIAN ALGORITMA *HEAP SORT*

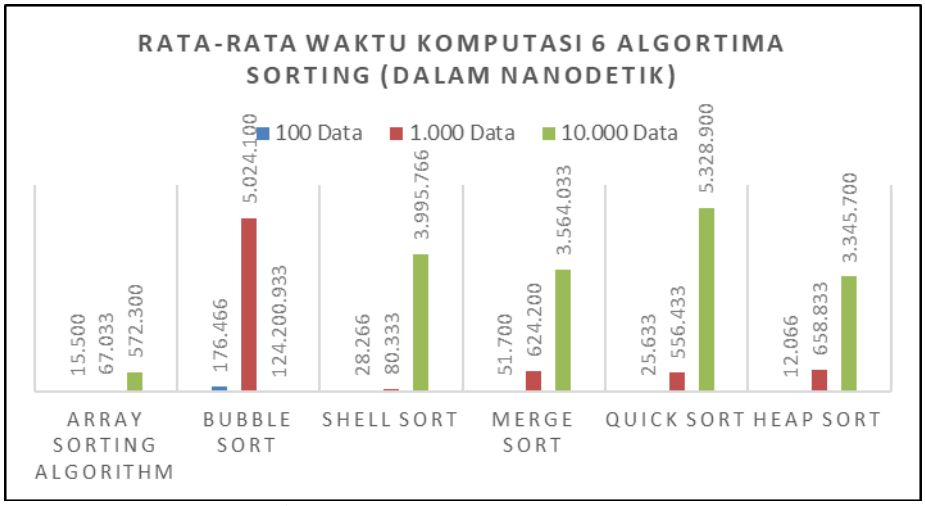
| Pengujian Ke | Jumlah Data | Pemakaian Memori (byte) | Waktu Komputasi (nanodetik) |
|---|---|---|---|
| 1 | 100 | 41.960 | 11.500 |
| 2 | 100 | 41.960 | 12.100 |
| 3 | 100 | 41.960 | 12.600 |
| 4 | 1.000 | 482.368 | 651.900 |
| 5 | 1.000 | 482.368 | 659.100 |
| 6 | 1.000 | 482.368 | 665.500 |
| 7 | 10.000 | 482.368 | 3.638.100 |
| 8 | 10.000 | 482.368 | 3.200.300 |
| 9 | 10.000 | 482.368 | 3.198.700 |

Berdasarkan Tabel 6, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 41.960 byte, dengan 1.000 data adalah 482.368 byte, dan dengan 10.000 data adalah 482.368 byte. Namun, dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 12.066 nanodetik, dengan 1.000 data adalah 658.833 nanodetik, dan dengan 10.000 data adalah 3.345.700 nanodetik. Dari hasil pengujian enam algoritma yang sudah dilakukan, rangkuman rata-rata hasil pengujian setiap algoritma bisa dilihat pada Gambar 2 dan 3 di bawah ini.




Gambar 2. Grafik Rata-Rata Pemakaian Memori Enam Algortima *Sorting*

Jurnal Manajemen Informatika (JAMIKA)
Volume 15 Nomor 1 Edisi April 2025
E ISSN: 2655-6960 | P ISSN: 2088-4125
OJS: https://ojs.unikom.ac.id/index.php/jamika




Gambar 3. Grafik Rata-Rata Waktu Komputasi Enam Algortima *Sorting*

Berdasarkan Gambar 2 dan 3 yang menampilkan rata-rata hasil pengujian dari enam algoritma *sorting* untuk tiga kategori jumlah data, yaitu 100, 1.000, dan 10.000, dapat diambil beberapa informasi sebagai berikut:

1. Untuk jumlah data 100, semua algoritma menunjukkan penggunaan memori yang sama, yaitu 41.960 byte. Waktu komputasi tercepat dimiliki oleh algoritma *Heap Sort* dengan rata-rata waktu 12.066 nanodetik, sedikit lebih cepat daripada algoritma ASA yang memiliki rata-rata waktu 15.500 nanodetik. Waktu komputasi paling lambat dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 176.466 nanodetik.
2. Ketika jumlah data meningkat menjadi 1.000, algoritma *Shell Sort* menunjukkan penggunaan memori yang lebih sedikit dibandingkan algoritma lainnya, yaitu 41.960 byte. Waktu komputasi tercepat dimiliki oleh algoritma ASA dengan rata-rata waktu 67.033 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 5.024.100 nanodetik.
3. Ketika jumlah data meningkat menjadi 10.000, algoritma *Merge Sort* menunjukkan penggunaan memori yang lebih besar dibandingkan algoritma lainnya, yaitu 965.976 byte, sedikit lebih banyak daripada algoritma ASA yang menggunakan 964.736 byte. Waktu komputasi tercepat dimiliki oleh algoritma ASA dengan rata-rata waktu 572.300 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh algoritma *Bubble Sort* dengan rata-rata waktu 124.200.933 nanodetik.

## IV. KESIMPULAN

Penelitian berhasil menguji dan membandingkan efisiensi waktu komputasi dan penggunaan memori antara *Array Sorting Algorithm* (ASA) dan lima algoritma pengurutan tradisional (*Bubble Sort*, *Shell Sort*, *Merge Sort*, *Quick Sort*, dan *Heap Sort*) dalam pengolahan data numerik acak menggunakan bahasa pemrograman Java. Hasil penelitian memperlihatkan bahwa ASA mengungguli algoritma pengurutan tradisional dalam hal kecepatan proses komputasi pada dataset 1.000 dan 10.000 data, serta berada di peringkat kedua pada dataset 100 data. Hal ini menegaskan efisiensi waktu komputasi yang telah diklaim. Namun, ASA juga memperlihatkan kelemahan dalam hal penggunaan memori yang lebih tinggi, khususnya pada dataset 10.000 data. Keunggulan ASA ini menunjukkan bahwa algoritma ini sangat cocok dalam skenario di mana waktu komputasi lebih diprioritaskan daripada penggunaan memori. Saran untuk pengembangan lebih lanjut mencakup penelitian pada pengoptimasian penggunaan memori ASA agar lebih efisien, serta evaluasi performa ASA dalam lingkungan pemrograman dan dataset yang berbeda untuk lebih memverifikasi keunggulan dan keterbatasannya algoritma ASA.

## DAFTAR PUSTAKA

[1] D. Awalludin, Y. Indrawan, dan R. Malfiany, "Pemodelan Sistem Informasi Pengelolaan Surat Pengantar Rujukan pada Rumah Sakit Menggunakan BPMN," *Jurnal Manajemen Informatika (JAMIKA)*, vol. 12, no. 2, hlm. 74–88, Sep 2022, doi: 10.34010/jamika.v12i2.7209.

[2] 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.

[3] 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.

[4] 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.

[5] D. Setyantoro dan R. A. Hasibuan, "ANALISIS DAN PERBANDINGAN KOMPLEKSITAS ALGORITMA EXCHANGE SORT DAN INSERTION SORT UNTUK PENGURUTAN DATA MENGGUNAKAN PYTHON," *Jurnal Ilmiah Teknik Informatika (TEKINFO)*, vol. 21, no. 1, hlm. 48–56, Apr 2020.

[6] H. Ali, H. Nawaz, A. Maitlo, dan I. Soomro, "Performance Analysis of Heap Sort and Insertion Sort Algorithm," *International Journal of Emerging Trends in Engineering Research*, vol. 9, no. 5, hlm. 580–586, Mei 2021, doi: 10.30534/ijeter/2021/08952021.

[7] 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.

[8] H. N. Elmahdy, "A New Sorting Algorithm for Integer Values (Array Sorting Algorithm)," dalam *The Proceedings of the Interdisciplinary Conference on Mechanics, Computers and Electrics (ICMECE 2022)*, Barcelona: ICMECE 2022, Okt 2022, hlm. 1–6.

[9] S. Saputri dan Y. Yahfizham, "Analisis Study Komperatif Bubble Sort Dan Selection Sort Pada Algoritma Dan Pemograman Berdasarkan Seleksi Pengurutan," *Jurnal Arjuna: Publikasi Ilmu Pendidikan, Bahasa dan Matematika*, vol. 1, no. 6, hlm. 151–161, Nov 2023.

[10] J. Jimmy, F. Utama, F. Felix, dan A. P. Laia, "Aplikasi Pembelajaran Penyortiran Menggunakan Algoritma Super Sort Berbasis Mobile," *Jurnal SIFO Mikroskil*, vol. 22, no. 1, hlm. 19–32, Agu 2021, doi: 10.55601/jsm.v22i1.771.

[11] 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.

[12] S. Sepahyar, R. Vaziri, dan M. Rezaei, "Comparing Four Important Sorting Algorithms Based on Their Time Complexity," dalam *Proceedings of the 2019 2nd International Conference on Algorithms, Computing and Artificial Intelligence*, New York, NY, USA: ACM, Des 2019, hlm. 320–327. doi: 10.1145/3377713.3377808.

[13] Z.-G. Zhu, "Analysis and Research of Sorting Algorithm in Data Structure Based on C Language," *J Phys Conf Ser*, vol. 1544, no. 1, hlm. 012002, Mei 2020, doi: 10.1088/1742-6596/1544/1/012002.

[14] D. A. Tara *dkk.*, "Analisis Kompleksitas Waktu Menggunakan Sorting Algorithm pada Pengaplikasian Fitur Pengurutan Harga dari Terendah dan Tertinggi di Shopee," *Jurnal Potensial*, vol. 3, no. 1, hlm. 68–80, Feb 2024.

[15] 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.

[16] A. Arrosyidi dan D. A. Arnandy, "Perbandingan Algoritma Simple Sorting Antara Menggunakan Variabel Temporary dan Tanpa Variabel Temporary," *Jurnal Sistim Informasi dan Teknologi*, vol. 4, no. 4, Des 2022, doi: 10.37034/jsisfotek.v4i4.185.

[17] 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.

[18] N. Mahrozi dan M. Faisal, "ANALISIS PERBANDINGAN KECEPATAN ALGORITMA SELECTION SORT DAN BUBBLE SORT," *Scientica: Jurnal Ilmiah Sains dan Teknologi*, vol. 1, no. 2, hlm. 89–98, Nov 2023.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] I. P. Pujiono, R. B. Trianto, dan F. M. Hana, "Perbandingan Efisiensi Memori dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java," *Jurnal Sistem Informasi dan Sistem Komputer*, vol. 9, no. 2, hlm. 2018–230, Jul 2024.