EFISIENSI MEMORI DAN WAKTU: ARRAY SORTING ALGORITHM VS ALGORTIMA PENGURUTAN TRADISIONAL MENGGUNAKAN PYTHON

Muhammad Zaki Musyaffa[1)], Karunia Raharjo[2)], Muhammad Faiz[3)], Satriaji Ammarullah[4)], Imam Prayogo Pujiono[5)]
Prodi Informatika, Fakultas Ekonomi & Bisnis Islam, UIN K.H.Abdurrahman Wahid Pekalongan

Correspondence author: M.Z.Musyaffa, muhammad.zaki.musyaffa24021@mhs.uingusdur.ac.id, Pekalongan, Indonesia

Abstract

The development of information technology has transformed data storage from physical to digital formats. However, digital data is difficult to access and verify without an effective sorting mechanism in place. This study compares the memory usage efficiency and execution time of the array sorting algorithm with two traditional sorting algorithms, bubble sort and quick sort, using a quantitative comparative method, all implemented in Python. Experiments were conducted on random numerical data sets of 100, 1.000, and 10.000 elements. Results show that the Array Sorting Algorithm excels in computational speed, with average times of 231.500, 352.500, and 1.214.150 nanoseconds for each data scale. However, it requires slightly larger memory (2.320 – 84.304 bytes). In contrast, Bubble Sort is the slowest but most memory-efficient, while Quick Sort is intermediate in both aspects. On the other hand, the Array Sorting Algorithm recorded relatively higher memory usage compared to the traditional sorting algorithms, Bubble Sort and Quick Sort. Based on these findings, algorithm selection should be based on the primary need. The Array Sorting Algorithm can be used when execution speed is a priority, and Bubble Sort is suitable for environments with memory constraints and small datasets. At the same time, Quick Sort offers a balance between speed and memory usage efficiency.

Keywords: array sorting algorithm, comparative, memory efficiency, execution time

Abstrak

Perkembangan teknologi informasi telah mentransformasi penyimpanan data dari format fisik ke digital. Namun, tanpa mekanisme pengurutan yang efektif, data digital sulit diakses dan diverifikasi. Penelitian ini membandingkan efisiensi penggunaan memori dan waktu eksekusi Array Sorting Algorithm dengan dua algoritma pengurutan tradisional, Bubble Sort dan Quick Sort menggunakan metode komparatif kuantitatif., semuanya diimplementasikan dalam bahasa Phyton. Eksperimen dilakukan pada kumpulan data numerik acak berukuran 100, 1.000, dan 10.000 elemen. Hasil menunjukkan bahwa Array Sorting Algorithm unggul dalam kecepatan komputasi dengan rata-rata waktu 231.500, 352.500, dan 1.214.150 nanodetik untuk masing-masing skala data, namun memerlukan memori sedikit lebih besar (2.320 – 84.304 byte), sedangkan Bubble Sort paling lambat namun paling hemat memori, dan Quick Sort menengah di kedua aspek.

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 05 NOMOR 02 JULI 2025
ISSN 2774 – 5775 | eISSN 2774-5767

Di sisi lain, Array Sorting Algorithm mencatatkan pemakaian memori yang relatif lebih tinggi dibanding algoritma pengurutan tradisional Bubble Sort dan Quick Sort. Berdasarkan hasil dari temuan ini pemilihan algoritma harus didasarkan pada kebutuhan utama, Array Sorting Algorithm bisa digunakan saat kecepatan eksekusi menjadi prioritas, Bubble Sort cocok untuk lingkungan dengan keterbatasan memori dan dataset berukuran kecil, sedangkan Quick Sort menawarkan keseimbangan antara kecepatan dan efisiensi penggunaan memori.

Kata Kunci: algoritma penguruan array, perbandingan, efisiensi memori, waktu eksekusi

A. PENDAHULUAN

Peranan teknologi informasi di zaman kontemporer sangatlah pesat perkembangannya membawa dampak ke dalam berbagai aspek kehidupan, seperti sistem komputasi dan penyimpanan data (Saputra & Purnawati, 2023). Jika dulu penyimpanan data dilakukan secara fisik dalam lemari arsip, kini telah beralih ke bentuk digital melalui perangkat komputer maupun *system cloud* (Tantowi & Wijayanti, 2023). Untuk mendukung pengelolaan data ini, berbagai algoritma pengurutan telah dirancang, baik untuk pengurutan secara menaik dan menurun (Nasution et al., 2023).

Bidang informatika didalamnya terdapat beberapa pendekatan utama yang digunakan dalam pembuatan algoritma pengurutan, seperti teknik *divide and conquer* (Sena dkk., 2024), pendekatan rekursif dan iteratif, serta algoritma berbasis perbandingan dan non perbandingan. Beberapa algoritma pengurutan antara lain *Bubble Sort* dan *Quick Short.* Algoritma *Quick Short* efektif dalam mengurutkan data besar, sedangkan *Bubble Short* efektif mengurutkan data kecil (D. S. Rita Wahyuni Arifin, 2020).

Efisiensi dalam penggunaan memori dan kecepatan komputasi merupakan dua aspek penting dalam mengevaluasi performa algoritma pengurutan (Astuti, 2023). Pemilihan algoritma yang tepat mampu menghemat sumber daya sekaligus mempercepat proses. Oleh sebab itu, pemahaman mendalam tentang berbagai algoritma menjadi sangat krusial (Farman et al., 2025). Salah satu inovasi terbaru dalam hal ini adalah *Array Sorting Algorithm,* yang dikembangkan untuk menyortir bilangan bulat positif secara efisien (Wijaya et al., 2024).

Array Sorting Algorihtm memanfaatkan array dua dimensi, dengan kolom pertama menyimpan nilai elemen dan kolom kedua mencatat frekuensinya (Pratama, 2020). Dengan mencatat jumlah kemunculan elemen, *Array Sorting Algorithm* dapat menyusun data secara lebih efisien. Kelebihan utama algoritma ini terletak pada efisiensi waktunya yang berkisar antara O(n) di kondisi terbaik dan O(2n) di kondisi terburuk, yang menjadikannya kompetitif dibandingkan algoritma pengurutan tradisional contohnya *Quick Sort* yang memiliki kompleksitas waktu O(n log n) dan *Bubble Sort* memiliki kompleksitas waktu $O(n^2)$ (Elmahdy, 2022). *Array Sorting Algorithm* dirancang khusus untuk bilangan bulat positif, yang banyak ditemukan pada data seperti nomor telepon dan NIM mahasiswa (Linawati et al., 2020).

Berbagai penelitian telah dilakukan untuk membandingkan efektivitas algoritma pengurutan dalam situasi tertentu. Beberapa studi menganalisis (Basir, 2020) performa *Merge Sort, Insertion Sort*, dan *Heap Sort* menggunakan Java, sementara studi lain (Tara dkk., 2024) menilai kecepatan algoritma dalam mengurutkan harga produk di *platform* Shopee. Penelitian (Pujiono et al., 2024; Sari et al., 2022) mengevaluasi kinerja

Bubble Sort, Shell Shot, dan *Quick Sort* menggunakan Java. Ada juga penelitian (Arrosyidi & Arnandy, 2022) yang membandingkan algoritma pengurutan dengan dan tanpa variabel sementara. Selain itu, studi tahun 2020 (D. S. Rita Wahyuni Arifin, 2020) meneliti efisiensi *Merge Sort* dan *Insertion Sort* menggunakan bahasa C++. Penelitian lain (Mahrozi & Faisal, 2023) mengevaluasi kecepatan antara *Bubble Sort* dan *Selection Sort* pada data berukuran 100 hingga 100.000 elemen. Di tahun 2022 (Poetra, 2022), *Quick Sort* dan *Bubble Sort* juga dibandingkan dalam konteks pengembangan aplikasi dengan Flutter dan Dart SDK.

Penelitian lainnya (Sandria dkk., 2022) mengeksplorasi algoritma *Selection Sort* dengan PHP, dan membandingkan *Insertion Sort, Selection Sort* dan *Bubble Sort* menggunakan Python (Iskandar et al., 2023).

Penelitian ini membandingkan *Array Sorting Algoritm* dengan dua algoritma pengurutan tradisional, yaitu *Bubble Sort* dan *Quick Sort*, dengan menggunakan Python (Ala'Anzy et al., 2024). Dataset berupa angka acak antara 1 sampai 99 digunakan dalam tiga skala berbeda: 100, 1.000, dan 10.000 elemen. Efisiensi memori diukur berdasarkan penggunaan memori oleh komputer, sedangkan efisiensi waktu dinilai dari lama proses pengurutan berlangsung. Performa *Array Sorting Algorithm* kemudian dianalisis dan dibandingkan untuk menilai apakah algoritma ini lebih unggul dalam hal efisiensi waktu dan memori dalam berbagai ukuran data (Pujiono et al., 2025).

B. METODE PENELITIAN

Dataset yang digunakan untuk klasifikasi Penelitian ini menggunakan array untuk membuat kumpulan data dalam tiga ukuran yang berbeda: 100 data berukuran kecil, 1.000 data berukuran sedang dan 10.000 data berukuran besar. Setiap kumpulan data mengandung angka numerik acak dengan nilai yang berkisar dari 1-99. Kode program digunakan untuk membuat array yang dapat digunakan untuk membuat dataset sebesar 100, 1.000, dan 10.000 dengan tipe data numerik acak yang memiliki nilai antara 1-99. Berikut ini contoh kode programnya:

```
import random
# Fungsi untuk membuat array
# Nantinya count diisi dengan angka 10000
def create_array(count):
 arr = [random.randint(1, 99) for _ in range(count)] # Membuat array dengan angka acak dari 1 sampai 99
 return arr # Mengembalikan array yang telah diisi

# Contoh penggunaan:
array = create_array(10000)
print(array[:10000]) # Menampilkan 10000 array
```

Dilihat dari kode program tersebut, mempunyai hasil yaitu array berjumlah 10.000 elemen. Kemudian, elemen-elemen ini dimasukkan dalam tiga variabel tipe data integer array, untuk melakukan pengujian terhadap *Array Sorting Algorithm, Quick Sort* dan *Bubble Sort* menggunakan data yang sama. Tiga variabel itu diantaranya: ArraySmall, ArrayMedium dan ArrayLarge. ArraySmall mempunyai isi 100 elemen pertama dari jumlah keseluruhan 10.000 elemen, ArrayMedium berisi 1.000 elemen pertama dan ArrayLarge berisi seluruh 10.000 elemen.

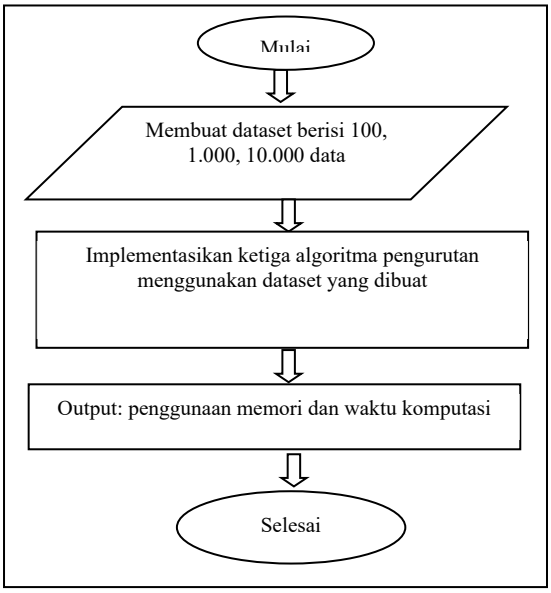
Berikut adalah *pseudocode* dari *Array Sorting Algorithm*:

1. Temukan nilai terbesar dalam array
2. Buat array 2 dimensi dengan ukuran [Largest +1,2]
3. Inisialisasi seluruh elemen array menjadi nol
4. Inisialisasi variabel penghitung ke nol
5. Masukkan elemen-elemen yang ingin diurutkan
6. Untuk setiap indeks i pada [Largest +1], masukkan elemen ke array dan tambahkan penghitung

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 05 NOMOR 02 JULI 2025
ISSN 2774 – 5775 | eISSN 2774-5767

7. Hapus elemen dengan penghitung nol
8. Tampilkan array

Jika tidak ada elemen yang berulang, dapat digunakan array satu dimensi tanpa penghitung. Hasil pengujian algoritma pengurutan, dalam Phyton, bisa di lihat sesuai Gambar 1.



Gambar 1. Alur Tahapan Penelitian

C. HASIL DAN PEMBAHASAN

Penelitian dan pengujian dilakukan menggunakan bahasa pemrograman Python dan *Integrated Development Environment* Visual Studio Code 1.89.1 untuk menjalankan program. Spesifikasi laptop untuk menjalankan pengujian ini sebagai berikut:

1. Prosesor AMD 3020e *with radeon graphics* dengan kecepatan 1.2 GHz
2. Memori 8GB RAM
3. Sistem Operasi Windows 11 64-Bit
4. SSD Berkapasitas 256 GB

Penelitian ini menguji *Array Sorting Algorithm*, *Bubble Sort* dan *Quick Sort* dengan dataset yang sama yang terdiri dari 100 data (ArraySmall), 1.000 data (ArrayMedium), dan 10.000 data (ArrayLarge) dengan angka acak antara 1 sampai 99. Setiap algoritma diuji sebanyak tiga kali:

1. Pengujian pertama sampai dengan kedua memakai 100 data dataset (ArraySmall)
2. Pengujian ketiga sampai dengan ke empat memakai 1.000 data dataset (ArrayMedium)
3. Pengujian kelima sampai dengan ke enam memakai 10.000 data dataset (ArrayLarge).

Pengujian dilakukan dua kali pada setiap kumpulan data untuk memastikan hasilnya konsisten dan meningkatkan kepercayaan terhadap hasilnya. Selama proses pengujian, hanya tiga aplikasi digunakan untuk memastikan bahwa aplikasi lain tidak mengganggu kinerja CPU dan memori. Untuk menjalankan ketiga program tersebut, Pertama, Visual Studio Code digunakan sebagai alat pengujian, kedua Microsoft Word yang digunakan untuk mencatat hasil tes dan ketiga Snipping Tools, yang dapat digunakan untuk mengambil tangkapan layar hasil tes.

Dokumentasi pengujian dataset program bisa di kunjungi pada tautan sebagai berikut: https://bit.ly/3GoUrsM. Setiap kali pengujian dilakukan, algoritma mempunyai parameter berdasarkan rentan waktu komputasi dan ukuran besaran pemakaian penggunaan memori. Berikut ini dokumentasi dari ketiga pengujian: tautan *Array Sorting Algorithm* (https://bit.ly/3Gu8o8U), tautan *Bubble Sorting* (https://bit.ly/44cJkx1) dan tautan *Quick Sort* (https://bit.ly/3RALkrm), lebih lanjut, penjelasan hasil pengujian akan diklasifikasikan lebih rinci.

Array Sorting Algorithm

```
#Fungsi utama ASA dalam bahasa pemrograman Python
def sorting_ASA(arr):
    if not arr:
        return []  # Jika array kosong, langsung kembalikan array kosong
    max_value = max(arr)  # Cari nilai maksimum dalam array
```

```
    count = [0] * (max_value + 1)  # Buat array count untuk menghitung frekuensi setiap angka

    for num in arr:
        count[num] += 1  # Hitung frekuensi kemunculan tiap angka

    sorted_arr = []
    for num, freq in enumerate(count):
        sorted_arr.extend([num] * freq)  # Bangun array terurut berdasarkan frekuensi

    return sorted_arr  # Kembalikan array yang sudah terurut
```

Salah satu Algoritma pengurutan data, *Array Sorting Algorithm* memiliki kesamaan dengan *Counting Sort*, diimplementasikan menggunakan bahasa pemrograman Python untuk menyusun elemen-elemen dalam sebuah array secara terurut. Pendekatan ini memanfaatkan penghitungan frekuensi kemunculan setiap elemen dalam array masukan untuk membentuk array keluaran yang terurut, tanpa memerlukan perbandingan langsung antar elemen. Fungsi inti, yakni sorting_ASA(arr), memulai proses dengan memverifikasi apakah array masukan kosong. Jika kondisi tersebut terpenuhi, fungsi mengembalikan array kosong untuk menangani kasus batas dengan efisien. Selanjutnya, nilai maksimum dari array ditentukan melalui fungsi bawaan Python max(arr) dan disimpan dalam variabel max_value. Nilai ini menjadi dasar untuk menginisialisasi array penghitung (count) sebagai array berisi nol dengan panjang max_value + 1. Array count digunakan untuk mencatat frekuensi kemunculan setiap elemen yang mungkin ada dalam array masukan, dengan indeks array merepresentasikan nilai elemen dan nilai pada indeks tersebut mencerminkan jumlah kemunculannya.

Proses penghitungan frekuensi dilakukan melalui iterasi pada array masukan menggunakan sintaks for num in arr, di mana setiap elemen num meningkatkan nilai pada posisi count[num]. Setelah frekuensi semua elemen tercatat, array terurut (sorted_arr) dibangun dengan mengiterasi array count menggunakan fungsi enumerate(count). Fungsi ini menghasilkan pasangan antara nilai elemen (num) dan frekuensinya (freq), yang kemudian dimanfaatkan untuk menambahkan elemen num sebanyak freq kali ke dalam sorted_arr melalui metode extend([num] * freq). Langkah ini menjamin bahwa elemen-elemen disusun sesuai urutan numeriknya dalam array keluaran. Fungsi akhirnya mengembalikan sorted_arr sebagai hasil dari proses pengurutan.

Algoritma ini menawarkan efisiensi yang signifikan, terutama pada dataset dengan rentang nilai terbatas atau elemen yang sering berulang, karena menghindari perbandingan langsung dan mengandalkan operasi penghitungan yang lebih cepat. Dalam implementasi ini, array count berfungsi sebagai struktur penyimpan frekuensi, dengan indeksnya secara implisit merepresentasikan elemen dan nilainya mencatat frekuensi kemunculan. Proses pengurutan diawali dengan identifikasi max_value untuk menetapkan ukuran array penghitung, dilanjutkan dengan penghitungan frekuensi, dan diakhiri dengan rekonstruksi array terurut berdasarkan data frekuensi. Pendekatan ini memastikan elemen-elemen berada pada posisi yang sesuai dengan kompleksitas waktu yang optimal untuk skenario tertentu, menjadikannya solusi yang efektif untuk data dengan karakteristik yang mendukung. Kode yang dihasilkan bersifat sederhana, mudah dipahami, dan secara akurat mencerminkan prinsip *Counting Sort* yang diadaptasi dalam *Array Sorting Algorithm.*

Data yang digunakan dalam penelitian ini akan diinput ke dalam program berdasarkan skenario pengujian yang telah dirancang sebelumnya. Pada tahap pengujian pertama hingga kedua, digunakan dataset berukuran kecil yang terdiri dari 100 elemen. Selanjutnya, pada pengujian ketiga hingga keempat, diterapkan dataset berukuran

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 05 NOMOR 02 JULI 2025
ISSN 2774 – 5775 | eISSN 2774-5767

menengah sebanyak 1.000 elemen. Sedangkan pada pengujian kelima hingga keenam, diuji dengan dataset berskala besar yang mencakup 10.000 elemen. Hasil dalam menjalankan uji program *Array Sorting Algoritm* sudah tercantum dalam Tabel 1, berikut ini tautan dokumentasi pengujian: https://bit.ly/3Gu8o8U.

Tabel 1. Hasil Pengujian *Array Sorting Algorithm*

Urutan Pengujian	Jumlah Data	Pemakaian Memori (*byte*)	Waktu Komputasi (nanodetik)
1	100	2.320	206.300
2	100	2.320	256.700
3	1.000	10.096	319.500
4	1.000	10.096	385.500
5	10.000	84.304	1.228.500
6	10.000	84.304	1.199.800

Berdasarkan Tabel 1, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 2.320 *byte*, dengan 1.000 data adalah 10.096 *byte*, dan dengan 10.000 data adalah 84.304 *byte*. Dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 231.500 nanodetik , dengan 1.000 data adalah 352.500 nanodetik, dan dengan 10.000 data adalah 1.214.150 nanodetik.

Bubble Sort

```
#Fungsi utama Bubble Sort dalam bahasa pemrograman Python
def bubble_sort(arr):
    n = len(arr)
    arr = arr.copy()  # Membuat salinan agar array asli tidak berubah
    for i in range(n):
        for j in range(0, n-i-1):
            if arr[j] > arr[j+1]:
                arr[j], arr[j+1] = arr[j+1], arr[j]  # Tukar jika elemen kiri lebih besar
    return arr  # Kembalikan array yang sudah terurut
```

Fungsi bubble_sort(arr) memulai proses. Langkah awal menentukan panjang array melalui sintaks n = len(arr). Salinan array dibuat menggunakan arr = arr.copy(), memastikan data asli tetap utuh. Proses pengurutan utama mengandalkan dua loop bersarang. Loop luar, ditulis sebagai for i in range(n), mengatur jumlah iterasi sesuai panjang array. Loop dalam, ditulis for j in range(0, n-i-1), membandingkan elemen pada indeks j dengan elemen berikutnya pada j+1. Jika elemen arr[j] melebihi arr[j+1], penukaran dilakukan melalui sintaks arr[j], arr[j+1] = arr[j+1], arr[j]. Setelah iterasi selesai, fungsi mengembalikan array terurut dengan perintah return arr. Intinya, setiap iterasi menggeser elemen terbesar menuju posisi akhir, menyerupai gelembung yang bergerak naik.

Implementasi kode dirancang intuitif, terorganisasi. Setiap putaran loop luar memosisikan elemen terbesar yang belum terurut ke tempat semestinya di akhir array. Bagian terurut bertambah satu elemen tiap iterasi. Batas loop dalam, n-i-1, diterapkan demi efisiensi, menghindari pemeriksaan ulang elemen yang sudah terurut. Proses perbandingan serta penukaran berlangsung hingga array sepenuhnya terurut, ditandai ketiadaan penukaran tambahan.

Bubble Sort menunjukkan kompleksitas waktu $O(n^2)$, kurang optimal untuk array besar. Meski demikian, algoritma ini unggul dalam kesederhanaan logika, cocok diterapkan pada data kecil atau kasus sederhana. Implementasinya memadukan efektivitas pengurutan dengan kemudahan pemahaman, menjadikannya pilihan praktis dalam konteks tertentu.

Data yang digunakan dalam penelitian ini akan diinput ke dalam program berdasarkan skenario pengujian yang telah dirancang sebelumnya. Pada tahap pengujian pertama hingga kedua, digunakan dataset berukuran kecil yang terdiri dari 100 elemen. Selanjutnya, pada pengujian ketiga hingga keempat, diterapkan dataset berukuran menengah sebanyak 1.000 elemen.

Sedangkan pada pengujian kelima hingga keenam, diuji dengan dataset berskala besar yang mencakup 10.000 elemen. Hasil dalam menjalankan uji program *Bubble Sort* sudah tercantum dalam Tabel 2, berikut ini tautan dokumentasi kode program: https://bit.ly/44cJkx1.

Tabel 2. Hasil Pengujian *Bubble Sort*

Urutan Pengujian	Jumlah Data	Pemakaian Memori (*byte*)	Waktu Komputasi (nanodetik)
1	100	2.256	1.219.200
2	100	2.256	1.121.600
3	1.000	9.456	1.210.209.200
4	1.000	9.456	1.081.204.200
5	10.000	81.456	148.428.761.400
6	10.000	81.456	143.996.802.300

Berdasarkan Tabel 2, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 2.256 *byte*, dengan 1.000 data adalah 9.456 *byte*, dan dengan 10.000 data adalah 81.456 *byte*. Dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 1.170.400 nanodetik, dengan 1.000 data adalah 1.145.706.700 nanodetik, dan dengan 10.000 data adalah 146.212.781.850 nanodetik.

Quick Sort

```
#Fungsi utama Quick Sort dalam bahasa pemrograman Python
def quick_sort(arr):
    if len(arr) <= 1:
        return arr  # Jika array berisi 0 atau 1 elemen, sudah terurut
    else:
        pivot = arr[len(arr) // 2]  # Ambil elemen tengah sebagai pivot
        left = [x for x in arr if x < pivot]    # Elemen lebih kecil dari pivot
        middle = [x for x in arr if x == pivot]  # Elemen sama dengan pivot
        right = [x for x in arr if x > pivot]    # Elemen lebih besar dari pivot
        return quick_sort(left) + middle + quick_sort(right)  # Rekursif urutkan kiri dan kanan
```

Kode Python yang diberikan mengimplementasikan algoritma *Quick Sort*, sebuah metode pengurutan efisien berbasis prinsip "*divide and conquer*". Fungsi quick_sort(arr) dimulai dengan memeriksa panjang array menggunakan len(arr) <= 1. Jika array memiliki 0 atau 1 elemen, fungsi langsung mengembalikan array tersebut karena sudah terurut. Namun, jika array lebih besar, elemen tengah dipilih sebagai pivot dengan sintaks pivot = arr[len(arr) // 2]. Pivot ini menjadi acuan untuk mempartisi array menjadi tiga bagian: left berisi elemen yang lebih kecil dari pivot, middle berisi elemen yang sama dengan pivot, dan right berisi elemen yang lebih besar dari pivot. Ketiga bagian ini dibentuk menggunakan list comprehension, seperti [x for x in arr if x < pivot] untuk left. Proses pengurutan dilanjutkan dengan memanggil fungsi quick_sort secara rekursif pada left dan right, lalu menggabungkan hasilnya dengan middle menggunakan operasi + dalam urutan quick_sort(left) + middle + quick_sort(right). Ini menghasilkan array yang terurut sepenuhnya.

Inti dalam struktur kode ini memecah array menjadi sub-array yang lebih kecil berdasarkan pivot, mengurutkan sub-array tersebut secara rekursif, dan menggabungkannya kembali untuk mendapatkan hasil akhir. Pemilihan elemen tengah sebagai pivot membantu membagi array secara merata, meskipun tidak selalu optimal untuk semua kasus. Penggunaan *list comprehension* membuat kode lebih ringkas dan mudah dibaca, sementara rekursi menjadi tulang punggung algoritma ini. Setiap iterasi memastikan elemen-elemen diposisikan dengan benar relatif terhadap pivot, sehingga kompleksitas waktu rata-rata *Quick Sort* menjadi O(n log n). Kode ini efektif untuk berbagai ukuran data, menunjukkan cara praktis menerapkan

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 05 NOMOR 02 JULI 2025
ISSN 2774 – 5775 | eISSN 2774-5767

konsep pengurutan dalam Python dengan kode program yang sederhana namun kuat.

Data yang digunakan dalam penelitian ini akan diinput ke dalam program berdasarkan skenario pengujian yang telah dirancang sebelumnya. Pada tahap pengujian pertama hingga kedua, digunakan dataset berukuran kecil yang terdiri dari 100 elemen. Selanjutnya, pada pengujian ketiga hingga keempat, diterapkan dataset berukuran menengah sebanyak 1.000 elemen. Sedangkan pada pengujian kelima hingga keenam, diuji dengan dataset berskala besar yang mencakup 10.000 elemen. Hasil dalam menjalankan uji program *Quick Sort* sudah tercantum dalam Tabel 3, berikut ini tautan dokumentasi pengujian: https://bit.ly/3RALkrm.

Tabel 3. Hasil Pengujian *Quick Sort*

Urutan Pengujian	Jumlah Data	Pemakaian Memori (*byte*)	Waktu Komputasi (nanodetik)
1	100	3.656	866.000
2	100	3.656	667.000
3	1.000	11.136	2.227.700
4	1.000	11.136	2.251.400
5	10.000	83.416	20.528.800
6	10.000	83.416	14.456.400

Berdasarkan Tabel 3, jika dilihat dari segi penggunaan memori, rata-rata pemakaian memori pada pengujian dengan 100 data adalah 3.656 *byte*, dengan 1.000 data adalah 11.136 *byte*, dan dengan 10.000 data adalah 83.416 *byte*. Dari perspektif waktu komputasi, waktu komputasi rata-rata untuk pengujian dengan 100 data adalah 766.500 nanodetik, dengan 1.000 data adalah 2.239.550 nanodetik, dan dengan 10.000 data adalah 17.492.600 nanodetik.

Tabel 4. Rata-Rata Pemakaian Memori 3 Algoritma Pengurutan

Jumlah Data	*Array Sorting Algorithm*	*Bubble Sort*	*Quick Sort*
100	2.320	2.256	3.656
1.000	10.096	9.456	11.136
10.000	84.304	81.456	83.416

Tabel 5. Rata-Rata Waktu Komputasi 3 Algoritma Pengurutan

Jumlah Data	*Array Sorting Algorithm*	*Bubble Sort*	*Quick Sort*
100	231.500	1.170.400	766.500
1.000	352.500	1.145.706.700	2.239.550
10.000	1.214.150	146.212.781.850	17.492.600

Dilihat dari Tabel 4 dan 5 dengan data yang menunjukkan rata-rata pengujian 3 algoritma pengurutan yang diklasifikasikan 3 jumlah data antara lain 100, 1.000 dan 10.000. Untuk jumlah data 100, semua algoritma menunjukkan penggunaan memori yang relatif kecil, berkisar antara 2.256 hingga 3.656 *byte*. Waktu komputasi tercepat dimiliki oleh *Array Sorting Algorithm* dengan rata-rata waktu 231.500 nanodetik, sedangkan waktu komputasi paling lambat dimiliki oleh *Bubble Sort* dengan rata-rata waktu 1.170.400 nanodetik.

Ketika jumlah data meningkat menjadi 1.000, penggunaan memori juga meningkat. *Quick Sor*t mencatat penggunaan memori tertinggi untuk kategori ini sebesar 11.136 *byte*, sedangkan penggunaan memori terendah tercatat pada *Bubble Sort* sebesar 9.456 *byte*. Dalam kategori ini, *Array Sorting Algorithm* tetap menunjukkan waktu komputasi tercepat dengan rata-rata 352.500 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh *Bubble Sort* dengan rata-rata 1.145.706.700 nanodetik.

Ketika jumlah data meningkat menjadi 10.000, semua algoritma mengalami peningkatan signifikan dalam penggunaan memori, dengan *Array Sorting Algorithm* mencatat penggunaan tertinggi sebesar 84.304 *byte*, sedangkan penggunaan memori terendah tercatat pada *Bubble Sort* sebesar 81.456 *byte*. Waktu komputasi tercepat tetap dimiliki oleh *Array Sorting Algorithm* dengan rata-rata waktu komputasi 1.214.150 nanodetik, sedangkan waktu komputasi paling lambat tetap dimiliki oleh *Bubble Sort* dengan rata-rata waktu 146.212.781.850 nanodetik.

D. PENUTUP

Penelitian ini berhasil menguji serta membandingkan efisiensi waktu proses dan pemakaian memori antara Array Sorting Algorithm dan 2 algoritma pengurutan tradisional Bubble Sort dan Quick Sort, dalam konteks pengolahan data numerik acak menggunakan Python. Hasil analisis menunjukkan bahwa ASA memiliki keunggulan dalam kecepatan komputasi pada seluruh skala data (100, 1.000, dan 10.000 data), dengan rata-rata waktu yang lebih cepat dibandingkan algoritma konvensional lainnya. Meski begitu, ASA menunjukkan penggunaan memori yang sedikit lebih besar, khususnya saat menangani dataset berukuran besar (10.000 data). Temuan ini menunjukkan bahwa Array Sorting Algorithm sangat sesuai diterapkan dalam situasi yang mengutamakan kecepatan pemrosesan dibandingkan efisiensi memori. Penelitian lanjutan sebaiknya meminimalkan konsumsi memori Array Sorting Algorithm serta menguji kinerjanya pada berbagai jenis data dan lingkungan pemrograman lain.

E. DAFTAR PUSTAKA

Ala'Anzy, M. A., Mazhit, Z., Ala'Anzy, A. F., Algarni, A., Akhmedov, R., & Bauyrzhan, A. (2024). Comparative Analysis of Sorting Algorithms: A Review. *2024 11th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, 88–100. https://doi.org/10.1109/ISCMI63661.2024.10851593

Arrosyidi, A., & Arnandy, D. A. (2022). Perbandingan Algoritma Simple Sorting Antara Menggunakan Variabel Temporary dan Tanpa Variabel Temporary. *Jurnal Sistim Informasi Dan Teknologi*. https://doi.org/10.37034/jsisfotek.v4i4.185

Astuti, Y. A. (2023). Analisis Pengujian Data Algoritma Bubble Sort. *REMIK: Riset Dan E-Jurnal Manajemen Informatika Komputer*, *7*(3), 1413–1420.

Basir, R. R. (2020). Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java. *STRING (Satuan Tulisan Riset Dan Inovasi Teknologi)*, *5*(2), 109. https://doi.org/10.30998/string.v5i2.6250

D. S. Rita Wahyuni Arifin. (2020). Algoritma Metode Pengurutan Bubble Sort dan Quick Dalam Bahasa Pemrograman C++. *Information System for Educators and Professionals*, *4*(2), 178–187.

Elmahdy, H. N. (2022). A New Sorting Algorithm for Integer Values (Array Sorting Algorithm). *The Proceedings of the Interdisciplinary Conference on Mechanics, Computers and Electrics (ICMECE 2022)*, *October*, 1–6.

Farman, I., Mukminin, M. A., Fitradani, N. Al, Fahmi, M., & Oktavia, O. (2025). Memori dan Media Penyimpanan. *JOET : Journal of Education and Technologies*, *1*(1), 16–27. https://jurnal.aaapublisher.com/index.php/JOET/article/view/33

Iskandar, J., Suhendar, H., & Pamungkas, B. D. (2023). Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python. *G-Tech: Jurnal Teknologi Terapan*, *8*(1), 104–113. https://doi.org/10.33379/gtech.v8i1.3556

Linawati, S., Nurdiani, S., Handayani, K., & Latifah, L. (2020). Prediksi Prestasi Akademik Mahasiswa Menggunakan Algoritma Random Forest dan C4.5. *Jurnal Khatulistiwa Informatika*, *8*(1). https://doi.org/10.31294/jki.v8i1.7827

Mahrozi, N., & Faisal, M. (2023). Analisis Perbandingan Kecepatan Algoritma

Selection Sort Dan Bubble Sort. *Jurnal Ilmiah Sain Dan Teknologi*, *1*(2), 89–98.

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

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

Pratama, M. A. (2020). *Struktur Data Array Dua Dimensi Pada Pemrograman C++*. https://doi.org/10.31219/osf.io/vyech

Pujiono, I. P., Rachmawanto, E. H., & Winarsih, N. A. S. (2025). Array Sorting Algorithm vs Traditional Sorting Algorithm: Memory and Time Efficiency Analysis. *Jurnal Manajemen Informatika (JAMIKA)*, *15*(1), 47–59. https://doi.org/10.34010/jamika.v15i1.13230

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

Sandria, Y. A., Nurhayoto, M. R. A., Ramadhani, L., Harefa, R. S., & Syahputra, A. (2022). Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP. *Hello World Jurnal Ilmu Komputer*, *1*(4), 190–194. https://doi.org/10.56211/helloworld.v1i4.187

Saputra, A. M. A., & Purnawati, L. P. I. K. A. A. R. M. I. B. N. W. (2023). *Teknologi Informasi: Peranan TI dalam berbagai bidang*. Jambi : PT. Sonpedia Publishing Indonesia.

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

Sena, M. B., Hanun, R. M., Purnama, I. R., Ardian, M., Studi, P., Informatika, T., & Pontianak, P. N. (2024). Perbandingan Kinerja Algoritma Sorting Dalam Pengurutan Data Menggunakan Bahasa Python. *Prosiding Seminar Nasional Sains Dan Teknologi Seri 02*, *1*(2), 310–314.

Tantowi, L., & Wijayanti, L. (2023). Peluang dan Tantangan Penyimpanan Cloud Storage Pada Dokumen Digital. *Shaut Al-Maktabah : Jurnal Perpustakaan, Arsip Dan Dokumentasi*, *15*(1), 118–131. https://doi.org/10.37108/shaut.v15i1.803

Tara, D. A., Ferdina, I., Sylvester, M. S., Firmansyah, M. F., Sulthonul Izza, M., Astuti, N. W., Putra Amarta, R., & Fajariansyah, R. (2024). Analisis Kompleksitas Waktu Menggunakan Sorting Algorithm pada Pengaplikasian Fitur Pengurutan Harga dari Terendah dan Tertinggi di Shopee. *Jurnal Potensial*, *3*(1), 68–80.

Wijaya, S., Fauziah, F., & Harjanti, T. W. (2024). Perbandingan Algoritma Sorting dengan Menggunakan Bahasa Pemograman Javascript dalam Penggunaan Waktu Komputasi dan Penggunaan Memori. *STRING (Satuan Tulisan Riset Dan Inovasi Teknologi)*, *8*(3), 294. https://doi.org/10.30998/string.v8i3.17972