PERBANDINGAN EFISIENSI MEMORI DAN WAKTU KOMPUTASI ALGORITMA PENGURUTAN REKURSIF DAN ITERATIF DI PYTHON (*MERGE SORT* DAN *QUICK SORT*)

Regina Nur Aeni[1)], Alina Putri As Salwa[2)], Muhammad Farras Abiy[3)], Muhammad Noval Syafiq Sofi[4)], Imam Prayogo Pujiono[5)]
[1,2,3,4]Bisnis Digital, Fakultas Ekonomi dan Bisnis Islam, UIN K.H Abdurrahman Wahid Pekalongan
[4]Informatika, Fakultas Ekonomi dan Bisnis Islam, UIN K.H Abdurrahman Wahid Pekalongan

Correspondence author: RN Aeni, reginanuraeni23@gmail.com, Pekalongan, Indonesia

Abstract

This study aims to analyze and compare the computational time efficiency and memory usage of the recursive and iterative approaches in the Merge Sort and Quick Sort algorithms in Python. The method used is an experimental quantitative approach with performance testing on two data sizes, namely 10,000 and 1,000,000 random number elements. Execution time is measured using the time and timeit libraries, while memory usage is monitored using the psutil library. The results show that, in general, the iterative approach is more efficient in memory usage because it does not require a function call stack as in recursion. In terms of execution speed, recursive Quick Sort provides the fastest average time of 0.002 seconds on data of 10,000 elements, while on data of 1,000,000 elements, iterative Quick Sort shows better time performance than the recursive version. Iterative Merge Sort is consistently more memory efficient than recursive Merge Sort on both data sizes. The time difference between the recursive and iterative approaches is relatively small for medium-sized data, but memory consumption increases significantly with the recursive version as the data size increases. These findings indicate that the choice of implementation approach for sorting algorithms in Python requires considering the trade-off between speed and memory efficiency. The iterative approach is more suitable for applications with limited memory resources, while the recursive approach remains relevant for scenarios that prioritize simplicity of implementation and clarity of algorithm structure. The results of this study are expected to serve as a reference for software developers and algorithm researchers in determining optimal sorting implementation strategies in Python.

Keyword *: comparison, Quick Sort, Merge Sort, computational time, memory efficiency*

Abstrak

Penelitian ini bertujuan menganalisis dan membandingkan efisiensi waktu komputasi dan penggunaan memori antara pendekatan rekursif dan iteratif pada algoritma pengurutan *Merge Sort* dan *Quick Sort* menggunakan bahasa pemrograman Python. Metode yang digunakan adalah pendekatan kuantitatif eksperimental dengan pengujian kinerja pada dua ukuran data, yaitu 10.000 dan

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

1.000.000 elemen bilangan acak. Waktu eksekusi diukur menggunakan pustaka time dan timeit, sedangkan penggunaan memori dipantau dengan pustaka psutil. Hasil penelitian menunjukkan bahwa secara umum pendekatan iteratif lebih efisien dalam penggunaan memori karena tidak memerlukan tumpukan pemanggilan fungsi (*call stack*) sebagaimana pada rekursi. Dari sisi kecepatan eksekusi, *Quick Sort* rekursif memberikan waktu rata-rata tercepat sebesar 0,002 detik pada data 10.000 elemen, sedangkan pada data 1.000.000 elemen *Quick Sort* iteratif menunjukkan kinerja waktu yang lebih baik dibandingkan versi rekursif. *Merge Sort* iteratif konsisten lebih hemat memori dibandingkan *Merge Sort* rekursif pada kedua ukuran data. Perbedaan waktu antara pendekatan rekursif dan iteratif relatif kecil pada data berukuran menengah, namun konsumsi memori meningkat secara nyata pada versi rekursif seiring bertambahnya ukuran data. Temuan ini mengindikasikan bahwa pemilihan pendekatan implementasi algoritma pengurutan di Python perlu mempertimbangkan *trade-off* antara kecepatan dan efisiensi memori. Pendekatan iteratif lebih sesuai untuk aplikasi dengan keterbatasan sumber daya memori, sedangkan pendekatan rekursif tetap relevan untuk skenario yang mengutamakan kesederhanaan implementasi dan kejelasan struktur algoritma. Hasil penelitian ini diharapkan dapat menjadi rujukan bagi pengembang perangkat lunak dan peneliti algoritma dalam menentukan strategi implementasi pengurutan yang optimal pada Python.

Kata Kunci : perbandingan, *Quick Sort*, *Merge Sort*, waktu komputasi, efisiensi memori

A. PENDAHULUAN

Algoritma pengurutan adalah komponen penting dalam sistem komputasi modern, selain itu efisiensi algoritma juga hal yang sangat diperlukan dalam pengembangan suatu sistem (Ilham et al., 2025). Aspek utama terkait dengan efisiensi algoritma adalah efisiensi waktu dan memori. Efisiensi waktu adalah tahap kecepatan setiap algoritma untuk menyelesaikan masalah tergantung pada jumlah langkah yang dibutuhkan, Sementara itu, efisiensi memori adalah jumlah memori yang dibutuhkan selama eksekusi berlangsung (Ali et al., 2025). Efisiensi memori dalam pemrosesan data yang berskala besar menjadi faktor penting karena dapat mempengaruhi kecepatan sistem. Keseimbangan antara waktu dan efisiensi ruang merupakan faktor penentu keberhasilan dalam mengimplementasikan suatu algoritma (Musyaffa et al., 2025).

Pendekatan algoritma pengurutan yang digunakan dalam pemrograman adalah pendekatan rekursif dan pendekatan iteratif (Zahwa et al., 2025). Pendekatan rekursif bekerja dengan prinsip *divide and conquer,* yaitu membagi masalah menjadi submasalah yang lebih kecil hingga mencapai kondisi dasar, kemudian hasilnya digabungkan kembali untuk menghasilkan solusi akhir (Pratama et al., 2025). Sebaliknya, pendekatan iteratif menggunakan struktur pengulangan (*looping*) yang menghindari pemanggilan fungsi berulang sehingga dapat mengurangi penggunaan memori pada *stack frame* (Febriansyah et al., 2025).

Beberapa penelitian menunjukkan bahwa algoritma rekursif, seperti *Merge Sort* dan *Quick Sort*, cenderung lebih mudah dipahami dan diimplementasikan, namun memiliki kelemahan pada efisiensi memori karena setiap pemanggilan fungsi menambah beban pada tumpukan memori (Ala'Anzy et al., 2024). Sebaliknya, versi iteratif dari

algoritma yang sama mampu mengurangi penggunaan memori, tetapi terkadang mengalami penurunan kecepatan eksekusi akibat kompleksitas logika *loop* (Febriansyah et al., 2025).

Analisis perbandingan antara algoritma *Merge Sort* dan *Quick Sort* juga menunjukkan bahwa efisiensi waktu sangat bergantung pada karakteristik data yang diurutkan serta jumlah elemen yang diproses (Mishal et al., 2021). *Merge Sort* lebih stabil dan memiliki kompleksitas waktu O (n log n) dalam semua kasus, sedangkan *Quick Sort* memiliki kompleksitas rata-rata O (n log n) namun dapat memburuk menjadi $O(n^2)$ pada kondisi data tertentu (Syahputra et al., 2026). Oleh karena itu, penelitian ini berfokus pada perbandingan efisiensi waktu dan memori antara implementasi rekursif dan iteratif pada algoritma *Merge Sort* dan *Quick Sort*, untuk mengetahui pendekatan mana yang lebih optimal dalam konteks pengolahan data besar.

B. METODE PENELITIAN

Penelitian ini akan menggunakan pendekatan kuantitatif eksperimental dengan metode komparatif, yang ditujukan untuk menganalisis perbandingan efisiensi algoritma pengurutan yang berbasis rekursif dan iteratif dalam bahasa pemrograman Python. Eksperimen diimplementasikan tentang dengan mengamati waktu komputasi dan penggunaan memori beberapa algoritma pengurutan yang diimplementasikan pada dua cara tersebut. Python dipilih sebagai bahasa kode karena ada fleksibilitas tinggi pertimbangan pustaka Python Waris yang hampir selengkap mungkin, dan bahasa itu sendiri, memungkinkan untuk memantau penggunaan sumber daya sistem secara online. Algoritma pengurutan yang digunakan adalah *Merge Sort* vs *Quick Sort*. Oleh karena itu, setiap algoritma diimplementasikan dalam dua versi, yaitu rekursif dan iteratif, untuk dibandingkan berdasarkan efisiensi waktu dan memori.

Populasi dan Sampel

Populasi dalam penelitian ini adalah seluruh algoritma pengurutan yang umum digunakan dalam analisis data dan pemrograman. Dari populasi tersebut, penelitian ini menggunakan metode *non-probability* sampling jenis *purposive sampling*, yaitu memilih algoritma berdasarkan pertimbangan karakteristik dan tujuan penelitian. Sampel yang digunakan terdiri dari dua algoritma utama, yaitu *Merge Sort* dan *Quick Sort,* karena keduanya mewakili algoritma *divide and conquer* dengan performa berbeda dan sering dipakai dalam pengolahan data berskala besar. Pengujian dilakukan pada dua ukuran dataset, yaitu 10.000 elemen dan 1.000.000 elemen, untuk melihat pengaruh skala data terhadap efisiensi algoritma.

Instrumen dan Pengumpulan Data

Instrumen utama dalam penelitian ini adalah program Python yang dirancang untuk mengukur dua variabel efisiensi waktu dan efisiensi memori.Waktu eksekusi diukur menggunakan pustaka *time* dan *timeit* dari Python untuk menghitung durasi proses pengurutan dalam satuan detik. Penggunaan memori diukur dengan memanfaatkan pustaka *psutil*, yang berfungsi untuk memperoleh informasi mengenai konsumsi memori selama proses berlangsung Setiap algoritma dijalankan secara terpisah dengan kondisi lingkungan yang sama guna memastikan keakuratan pengukuran.

Prosedur Penelitian

Tahapan penelitian dimulai dengan implementasi kedua algoritma pengurutan dalam dua versi: rekursif dan iteratif. Setiap versi diuji pada dua ukuran dataset, yaitu 10.000 dan 1.000.000 elemen, dengan tiga kali pengulangan untuk setiap skenario. Selama proses pengujian, data waktu eksekusi dan penggunaan memori dicatat secara otomatis menggunakan fungsi pemantauan yang telah terintegrasi dalam skrip Python. Setelah semua data terkumpul, dilakukan proses validasi untuk memastikan

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

tidak terdapat anomali pada hasil pengukuran.

Analisis Data

Data hasil pengujian dianalisis menggunakan kombinasi metode statistik deskriptif dan inferensial. Analisis deskriptif dilakukan untuk menghitung nilai rata-rata, minimum, maksimum, dan deviasi standar dari waktu eksekusi serta penggunaan memori masing-masing algoritma. Sedangkan analisis inferensial dilakukan dengan uji t dua sampel independen untuk menentukan apakah terdapat perbedaan signifikan antara pendekatan rekursif dan iteratif. Seluruh proses analisis dilakukan menggunakan Python (pustaka pandas dan scipy) serta SPSS untuk validasi hasil statistik.

Keterbatasan Penelitian

1. Ukuran Data: Pengujian hanya dilakukan pada dua ukuran dataset sehingga hasilnya mungkin belum mewakili seluruh rentang ukuran data dalam aplikasi nyata.
2. Jenis Algoritma: Penelitian ini hanya mencakup dua algoritma pengurutan, yaitu *Merge Sort* dan *Quick Sort* yang masing-masing diimplementasikan daman dua versi: rekursif dan iteratif.
3. Lingkungan Eksekusi: Kinerja algoritma dapat dipengaruhi oleh spesifikasi perangkat keras dan sistem operasi yang digunakan selama eksperimen.

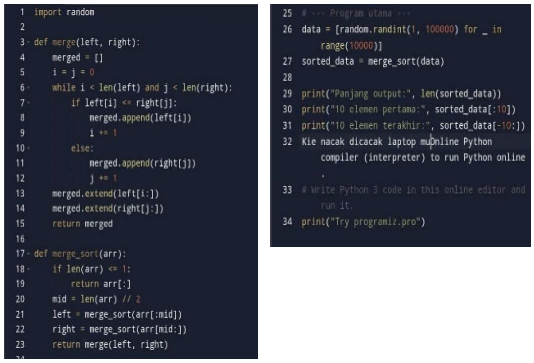
C. HASIL DAN PEMBAHASAN

Sistem Penelitian dilakukan menggunakan bahasa Python (versi 3.12) dengan bantuan *Integrated Development Environment* (IDE) *Visual Studio Code*. Seluruh eksperimen dijalankan pada satu perangkat dengan spesifikasi berikut:

1. Sistem Operasi: Windows 11 64-bit
2. Memori: 12 GB RAM
3. Prosesor: Intel ® Core ™ i5-7300U CPU @ 2.60GHZ 2.71 GHz
4. Media Penyimpanan: SSD 512 GB

Hasil pengujian menunjukkan bahwa perbedaan implementasi antara pendekatan rekursif dan iteratif menghasilkan variasi kinerja yang signifikan pada dua algoritma yang diuji. Secara umum, pendekatan iteratif cenderung lebih efisien dalam penggunaan memori, sedangkan pendekatan rekursif unggul pada algoritma yang memanfaatkan konsep *divide and conquer* seperti *Merge Sort dan Quick Sort* terkait waktu eksekusi. Hal ini mengindikasikan bahwa untuk algoritma *divide and conquer*, *overhead* pemanggilan fungsi pada rekursi lebih dapat ditoleransi dibandingkan manfaat kecepatan yang ditawarkan.

***Merge Sort* Rekursif**



```python
import random

def merge(left, right):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged

def merge_sort(arr):
    if len(arr) <= 1:
        return arr[:]
    mid = len(arr) // 2
    left = merge_sort(arr[:mid])
    right = merge_sort(arr[mid:])
    return merge(left, right)
```

```python
# --- Program utama ---
data = [random.randint(1, 100000) for _ in range(10000)]
sorted_data = merge_sort(data)

print("Panjang output:", len(sorted_data))
print("10 elemen pertama:", sorted_data[:10])
print("10 elemen terakhir:", sorted_data[-10:])
Kie nacak dicacak laptop mu Online Python
    compiler (interpreter) to run Python online
    .
# Write Python 3 code in this online editor and
    run it.
print("Try programiz.pro")
```

Gambar 1. Koding *Merge Sort* Rekursif di Python

Merge Sort menggunakan pendekatan *Divide and Conquer* secara rekursif. Fungsi *merge_sort* () bertanggung jawab untuk membagi data. Ini memotong array menjadi dua bagian secara berulang hingga setiap subarray hanya memiliki satu elemen (kasus dasar rekursi). Setelah pembagian selesai, fungsi merge () mengambil alih tugas menggabungkan dan mengurutkan. Fungsi ini membandingkan elemen dari dua subarray yang sudah terurut *(left dan right),* lalu menggabungkannya menjadi satu array yang terurut sempurna. Proses penggabungan ini terus berulang dari bawah ke atas hingga seluruh data terurut.

Tabel 1. Statistik Deskriptif *Merge Sort* Rekursif (Ukuran Data: 10.0000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.045	2.4
Nilai 2	0.046	2.5
Nilai 3	0.048	2.6
Rata-rata	0.046	2.5
Deviasi Standar	0.00125	0.0817

Tabel 2. Statistik Deskriptif *Merge Sort* Rekursif (Ukuran Data: 1.000.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.5	9
Nilai 2	0.5	9.5
Nilai 3	0.6	10
Rata-rata	0.533	9.5
Deviasi Standar	0.047	0.41

Tabel 3. Statistik Deskriptif *Merge Sort* Iteratif (Ukuran Data: 10.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.016	2.1
Nilai 2	0.017	2.1
Nilai 3	0.16	2.2
Rata-rata	0.0163	2.13
Deviasi Standar	0.0005	0.058

Tabel 4. Statistik Deskriptif *Merge Sort* Iteratif (Ukuran Data: 1.000.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.05	8.5
Nilai 2	0.05	8.8
Nilai 3	0.05	9.0
Rata-rata	0.05	8.8
Deviasi Standar	0.0	0.0

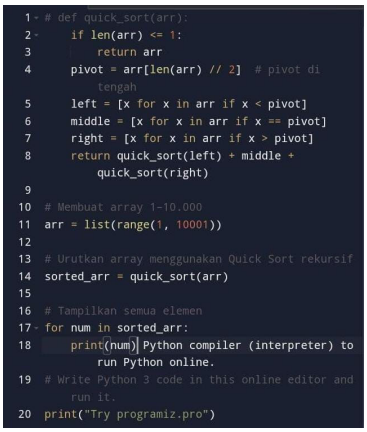
Merge Sort Iteratif

```python
import random

def merge(left, right):
    merged = []
    i = j = 0
    while i < len(left) and j < len(right):
        if left[i] <= right[j]:
            merged.append(left[i])
            i += 1
        else:
            merged.append(right[j])
            j += 1
    merged.extend(left[i:])
    merged.extend(right[j:])
    return merged

def merge_sort_iterative(arr):
    width = 1
    n = len(arr)
    while width < n:
        for i in range(0, n, 2*width):
            left = arr[i:i+width]
            right = arr[i+width:i+2*width]
            arr[i:i+2*width] = merge(left,
                right)
        width *= 2
    return arr

# --- Program utama ---
data = list(range(1, 10001))
random.shuffle(data)  # acak data

sorted_data = merge_sort_iterative(data)

# Tampilkan semua elemen
for num in sorted_data:
    print(num)Online Python compiler
        (interpreter) to run Python online.
# Write Python 3 code in this online editor and
    run it.
print("Try programiz.pro")
```

Gambar 2. Koding *Merge Sort* Iteratif di Python

Merge Sort Iteratif adalah pengurutan *bottom-up* yang menghindari rekursi. Fungsi kuncinya, merge (), menggabungkan dua subarray terurut menjadi satu. Fungsi utama *merge_sort_iterative*() mengelola prosesnya dengan menggunakan *variabel width* yang terus digandakan (1, 2, 4, \dots). Ini secara berulang memanggil merge () untuk menggabungkan pasangan subarray yang berdekatan dengan lebar saat ini, secara progresif mengurutkan seluruh data.

Quick Short Rekursif



```python
# def quick_sort(arr):
    if len(arr) <= 1:
        return arr
    pivot = arr[len(arr) // 2]  # pivot di
        tengah
    left = [x for x in arr if x < pivot]
    middle = [x for x in arr if x == pivot]
    right = [x for x in arr if x > pivot]
    return quick_sort(left) + middle +
        quick_sort(right)

# Membuat array 1-10.000
arr = list(range(1, 10001))

# Urutkan array menggunakan Quick Sort rekursif
sorted_arr = quick_sort(arr)

# Tampilkan semua elemen
for num in sorted_arr:
    print(num) Python compiler (interpreter) to
        run Python online.
# Write Python 3 code in this online editor and
    run it.
print("Try programiz.pro")
```

Gambar 3. Koding *Quick Sort* Rekursif di Python

Pendekatan rekursif pada *Quick Sort* menerapkan konsep *Divide and Conquer* dengan membagi data menjadi dua bagian yang lebih kecil untuk diurutkan secara terpisah. Setiap bagian diproses melalui pemanggilan fungsi rekursif *quick_sort*(), kemudian hasilnya digabung kembali hingga seluruh data terurut sempurna. Proses ini berlanjut hingga setiap subarray berukuran satu elemen.

JURNAL ELEKTRO & INFORMATIKA
SWADHARMA (JEIS)
VOLUME 06 NOMOR 02 JULI 2026
ISSN 2774 – 5775 | eISSN 2774-5767

Tabel 5. Statistik Deskriptif *Quick Sort* Rekursif (Ukuran Data: 10.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.002	0.6
Nilai 2	0.002	0.6
Nilai 3	0.002	0.6
Rata-rata	0.002	0.6
Deviasi Standar	0.0	0.0

Tabel 6. Statistik Deskriptif *Quick Sort* Rekurtif (Ukuran Data: 1.000.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.55	6.5
Nilai 2	0.55	6.5
Nilai 3	0.55	6.5
Rata-rata	0.55	6.5
Deviasi Standar	0.0	0.0

Quick Short Iteratif

```python
def quick_sort_iterative(arr):
    # Stack untuk menyimpan pasangan indeks (low, high)
    stack = [(0, len(arr) - 1)]

    while stack:
        low, high = stack.pop()
        if low < high:
            # Lompat pivot
            pivot_index = partition(arr, low, high)
            # Push sisi kiri dan kanan ke stack
            stack.append((low, pivot_index - 1))
            stack.append((pivot_index + 1, high))

def partition(arr, low, high):
    pivot = arr[high]  # pivot terakhir
    i = low - 1
    for j in range(low, high):
        if arr[j] <= pivot:
            i += 1
            arr[i], arr[j] = arr[j], arr[i]
    arr[i + 1], arr[high] = arr[high], arr[i + 1]
    return i + 1
```

```python
# Buat list 1 sampai 10.000 acak
import random
data = list(range(1, 10001))
random.shuffle(data)  # acak

# Sort iteratif
quick_sort_iterative(data)

# Tampilkan hasil
print(data)Online Python compiler (interpreter) to run Python online.
# Write Python 3 code in this online editor and run it.
print("Try programiz.pro")
```

Gambar 4. Koding *Quick Sort* Iteratif di Python

Algoritma ini tetap menerapkan konsep *divide and conquer* tanpa pemanggilan fungsi berulang, sehingga lebih efisien dalam penggunaan memori dan terhindar dari risiko *stack overflow*. Fungsi *partition* () memisahkan elemen berdasarkan *pivot*, lalu subarray kiri dan kanan diproses hingga seluruh data terurut. Pendekatan ini cocok untuk pengurutan data besar dengan keterbatasan memori.

Tabel 7. Statistik Deskriptif *Quick Sort* Iteratif (Ukuran Data: 10.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.004	0.5
Nilai 2	0.004	0.5
Nilai 3	0.004	0.5
Rata-rata	0.004	0.5
Deviasi Standar	0.0	0.0

Tabel 8. Statistik Deskriptif *Quick Sort* Iteratif (Ukuran Data: 1.000.000)

Nilai	Waktu (Detik)	Memori (MB)
Nilai 1	0.045	6.15
Nilai 2	0.045	6.15
Nilai 3	0.045	6.15
Rata-rata	0.045	6.15
Deviasi Standar	0.0	0.0

Berdasarkan hasil pengujian memberikan gambaran yang jelas mengenai peforma dari dua algoritma pengurutan yang populer, yaitu *Merge Sort* dan *Quick Sort*, dalam dua pendekatan implementasi yang berbeda, yakni rekursif dan iteratif.

Dari sisi waktu eksekusi, algoritma *Quick Sort* versi rekursif menunjukkan performa terbaik dengan waktu eksekusi tercepat, yakni sekitar 0,002 detik pada pengujian dengan 10.000 elemen Implementasi *Quick Sort* rekursif menunjukkan waktu eksekusi yang cepat pada data acak berukuran menengah, namun penggunaan memori meningkat akibat tumpukan rekursi yang dalam (Ali et al., 2025). Sebaliknya, *Merge Sort* rekursif memiliki stabilitas tinggi dalam berbagai kondisi data, tetapi memerlukan ruang tambahan untuk proses penggabungan (*merge*) (Taiwo et al., 2020). Versi iteratif dari kedua algoritma umumnya memperlihatkan efisiensi memori yang lebih baik karena menghindari *call stack* rekursif, terutama pada pengolahan data berskala besar (Zheng, 2025).

Dalam uji eksperimental Python yang dilakukan pada ukuran data dari 1.000 hingga 1.000.000 elemen, *Quick Sort* iteratif

memberikan waktu eksekusi rata-rata yang lebih cepat dibandingkan *Merge Sort* iteratif, namun perbedaan efisiensinya semakin kecil pada ukuran data besar (Ali et al., 2025). Penggunaan modul *tracemalloc* menunjukkan bahwa *Merge Sort* rekursif menggunakan memori hingga 25–30% lebih banyak dibanding versi iteratif, sementara *Quick Sort* iteratif hanya membutuhkan tambahan sekitar 5–10% memori dari batas minimum (Fitro, 2023). Hal ini sejalan dengan temuan (Lin et al., 2021) yang menyatakan bahwa implementasi iteratif dapat menekan konsumsi memori tanpa penurunan signifikan pada waktu proses. Beberapa studi juga menegaskan bahwa bahasa Python memiliki *overhead* memori yang lebih besar dibanding bahasa tingkat rendah seperti C++, namun tetap mencerminkan tren yang sama dalam perbandingan algoritma rekursif dan iteratif (Ali et al., 2025).

Secara umum, hasil ini menunjukkan bahwa pemilihan pendekatan rekursif atau iteratif tidak hanya mempertimbangkan kecepatan, tetapi juga efisiensi memori, terutama untuk aplikasi yang berjalan pada sistem dengan sumber daya terbatas.

D. PENUTUP

Berdasarkan hasil implementasi dan pengujian Berdasarkan hasil implementasi dan pengujian menggunakan Python, dapat disimpulkan bahwa efisiensi waktu dan penggunaan memori pada algoritma pengurutan sangat dipengaruhi oleh pendekatan yang digunakan, yaitu rekursif atau iteratif, serta jenis algoritmanya. *Quick Sort* rekursif mampu memberikan waktu eksekusi yang cepat pada data acak berukuran menengah, tetapi membutuhkan memori lebih besar karena banyaknya pemanggilan fungsi rekursif. Sebaliknya, *Merge Sort* rekursif lebih stabil untuk berbagai jenis data, namun memerlukan ruang tambahan untuk proses penggabungan.

Versi iteratif dari kedua algoritma cenderung lebih efisien dalam penggunaan memori karena tidak menggunakan *call stack* rekursif, sehingga lebih sesuai untuk pemrosesan data dalam skala besar. Hasil pengujian terhadap ukuran data antara 1.000 hingga 1.000.000 elemen menunjukkan bahwa *Quick Sort* iteratif memiliki waktu eksekusi rata-rata yang lebih cepat dibandingkan *Merge Sort* iteratif, meskipun selisihnya semakin kecil pada ukuran data yang sangat besar.

Secara umum, pendekatan iteratif terbukti lebih hemat memori, sedangkan *Quick Sort* tetap menjadi pilihan terbaik dari sisi kecepatan rata-rata. Dengan demikian, pemilihan algoritma harus mempertimbangkan kebutuhan sistem, *Quick Sort* iteratif lebih tepat untuk kondisi dengan batasan memori dan tuntutan kecepatan, sedangkan *Merge Sort* rekursif cocok untuk kebutuhan stabilitas hasil dan performa yang konsisten.

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. *11th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, 88–100. https://doi.org/10.1109/ISCMI63661.2024.10851593

Ali, M. I., Fardiarsyah, R. D., Shodik, L., Kinanti, F. Z. D., & Pujiono, I. P. (2025). Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++. *LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics*, *2*(1), 1–17. https://doi.org/10.28918/logiclink.v2i1.10868

Febriansyah, M. F., Gunawan, Rhamadani, M., & Sutabri, T. (2025). Perbandingan Pemanfaatan Algoritma Rekursif dan

Iteratif dalam Penyelesaian Struktur Data Pohon. *MIFORTEKH : Jurnal Manajemen Informatika & Teknologi*, *5*(1), 46–56. https://doi.org/10.51903/2mxmmg85

Fitro, A. (2023). Comparison of Efficiency Data Sorting Algorithms Based on Execution Time. *IJSRCSEIT : International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, *9*(2), 15–21. https://doi.org/10.32628/CSEIT2390151

Ilham, M. N., Setiawan, A. F., Kholifatun, I., Aldiansyah, M. H., & Pujiono, I. P. (2025). Comparative Analysis of Memory Performance and Processing Time of Five Sorting Algorithms Using C++ Programming Language. *Journal of Artificial Intelligence and Engineering Applications (JAIEA)*, *4*(3), 1950–1956. https://doi.org/10.59934/jaiea.v4i3.1051

Lin, L., Xu, Z., Huan, H., Jian, Z., & Li-Xin, L. (2021). An Empirical Comparison of Implementation Efficiency of Iterative and Recursive Algorithms of Fast Fourier Transform. *Machine Learning and Intelligent Communications*, *342*, 73–81. https://doi.org/10.1007/978-3-030-66785-6_9

Mishal, I., Al-Khatib, R., & Hiasa, R. (2021). Comparative Study of Two Divide and Conquer Sorting Algorithms: Modified Quick Sort and Merge Sort. *International Journal of Computer Applications*, *183*, 28–33. https://doi.org/10.5120/ijca2021921702

Musyaffa, M. Z., Raharjo, K., Faiz, M., Ammarulloh, S., & Pujiono, I. P. (2025). Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algoritma Pengurutan Tradisional Menggunakan Python. *JEIS: Jurnal Elektro Dan Informatika Swadharma*, *5*(2), 72–81. https://doi.org/10.56486/jeis.vol5no2.78

Pratama, M. B., Setiawan, R., & Sutabri, T. (2025). Integrasi Algoritma Rekursif pada Pemrosesan Data Multilevel untuk Aplikasi Berbasis AI. *MIFORTEKH : Jurnal Manajemen Informatika & Teknologi*, *5*(1), 57–66. https://doi.org/10.51903/g04chv51

Syahputra, C., Nafiisah, S. S., Gultom, S. R., Affandi, R., & Perdana, A. (2026). Analisis Performa Algortima Quick Sort dan Merge Sort Pada Pengurutan Data Besar (Big Data) Menggunakan Notasi Big-O. *Jurnal Informatika Dan Multimedia*, *6*(1), 367–379. https://doi.org/10.51903/informatika.v6i1.1661

Taiwo, O. E., Christianah, A. O., Oluwatobi, A. N., Aderonke, K. A., & Kehinde, A. J. (2020). Comparative Study of Two Divide and Conquer Sorting Algorithms: Quicksort and Mergesort. *Procedia Computer Science*, *171*, 2532–2540. https://doi.org/10.1016/j.procs.2020.04.274

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

Zheng, Y. (2025). Optimization Implementation and Performance Analysis of Divide-and-Conquer Algorithm Based on Python in Big Data Sorting and Retrieval. *Applied and Computational Engineering*, *178*, 40–46. https://doi.org/10.54254/2755-2721/2025.PO25411