Analisis Kompleksitas Waktu pada Algoritma Merge Sort

April Hardinata (103022300027), Emillio Syailendra wijaya (103022330124)

School of Computing

Telkom University

Bandung, Indonesia

Abstract - Makalah ini menyajikan analisis tentang kompleksitas waktu algoritma *merge sort recursive* dan juga algoritma merge sort iterative Algoritma ini bekerja dengan membagi larik masukan secara rekursif menjadi bagian yang lebih kecil, mengurutkan bagian-bagian tersebut, dan kemudian menggabungkan kembali bagian-bagian tersebut untuk menghasilkan larik yang telah diurutkan.

I. INTRODUCTION

Ada banyak pengaplikasian algoritma *Sorting* dalam ilmu komputer, mulai dari analisis data hingga manajemen basis data. *Merge Sort* menonjol di antara metode pengurutan karena pendekatan *divide-and-conquer* dan kinerjanya yang terjamin setiap kali. Merge Sort, yang diciptakan oleh John von Neumann pada tahun 1945, beroperasi dengan membagi *array* menjadi *subarray* yang lebih kecil, mengurutkan setiap *subarray*, dan kemudian menggabungkannya kembali menjadi array yang terurut. Algoritma tidak cukup hanya benar tetapi juga harus efisien. terdapat beberapa cara bagaimana kita bisa mengukur sebuah algoritma itu benar dan efisien.
Berikut adalah cara-cara kita mengukur suatu algoritma

1. Evaluasi Kompleksitas
   Untuk mengukur efisiensi algoritma tertentu, bisa dilakukan dengan mengukur waktu jumlah operasi dasar yang dilakukan algoritma berdasarkan ukuran input (n). Analisis dilakukan menggunakan menggunakan notasi asimtotik seperti O (Big Oh), Ω (Big Omega), dan Θ (Big Theta).
   - Kompleksitas Waktu:
     waktu algoritma dianalisis berdasarkan 3 scenario: *best-case* atau biasa di lambangkan Ω, *worst-case* dilambangkan O dan *average-case* Θ.

2. Evaluasi Kebenaran
   Pendekatan formal, yang mencakup analisis logika dan bukti matematis, digunakan untuk memastikan bahwa algoritma ini benar baik untuk kasus dasar maupun langkah rekursifnya. Ini dilakukan dengan metode induksi matematis. Untuk memastikan bahwa algoritma dapat menghasilkan hasil yang konsisten dan akurat, berbagai jenis data diuji secara menyeluruh, termasuk *worst-case, best-case, average-case*.

   - Pada langkah rekursif, algoritma menggunakan pendekatan induksi untuk memastikan bahwa setiap subarray yang dibuat akan terurut dengan benar sebelum digabungkan kembali ke dalam array utama

II. ASYMPTOTIC TIME COMPLEXITY

A. Kompleksitas Algoritma

untuk menangani suatu masalah, ada banyak algoritma yang dapat digunakan. Namun, penting untuk mengetahui algoritma mana yang paling efektif untuk menyelesaikan masalah tersebut. Tentu saja, untuk

membandingkan algoritma-algoritma tersebut harus memenuhi beberapa kriteria.

B. Notasi Asimtotik
Penghitungan waktu berjalan dari setiap bagian kode atau operasi dalam unit matematika dari suatu komputasi disebut analisis asimtotik. Dihitung sebagai fungsi, misalnya $f(n)$. Analisis asimtotik, juga dikenal sebagai asimtotik, adalah teknik dalam analisis matematika yang menggambarkan perilaku yang membatasi.

Tiga jenis waktu yang dibutuhkan oleh algoritma berbeda-beda. Waktu terburuk adalah waktu yang paling banyak digunakan atau dilakukan oleh algoritma saat menganalisisnya. Waktu terbaik adalah waktu minimum yang dibutuhkan oleh algoritma atau bagian kode dan biasanya tidak dihitung saat menganalisisnya. Waktu rata-rata adalah waktu rata-rata yang dibutuhkan oleh algoritma atau bagian kode dan terkadang dilakukan saat menganalisis algoritma.

- Big Oh, O
  Digunakan untuk mengukur kinerja atau kompleksitas suatu algoritma. Dalam istilah yang lebih matematis, ini adalah batas atas laju pertumbuhan suatu fungsi, atau jika suatu fungsi g(x) tumbuh tidak lebih cepat daripada suatu fungsi f(x), maka g disebut sebagai anggota O(f). Secara umum, ini digunakan untuk menunjukkan batas atas suatu algoritma dan memberikan ukuran untuk kompleksitas waktu terburuk atau waktu terlama yang mungkin (Worst-case)

- Big Omega, Ω
  Digunakan untuk menunjukkan batas bawah waktu berjalan suatu algoritma. Notasi ini mengukur kompleksitas waktu kasus terbaik atau jumlah waktu terbaik(Best-case) yang mungkin diperlukan algoritma untuk menyelesaikannya.

- Big Theta, Θ
  Digunakan untuk mengukur batas atas dan batas bawah (Average-case)

C. Kelas Efisiensi
Kompleksitas waktu asimptotik, yang ditunjukkan dengan "O", berasal dari term terbesar dalam persamaan kompleksitas waktu.

Kelompok	Nama
$O(1)$	konstan
$O(log\ n)$	logaritmik
$O(n)$	linier
$O(n\ log\ n)$	n log n
$O(n^2)$	kuadratik
$O(n^3)$	kubik
$O(2^n)$	eksponensial
$O(n!)$	faktorial

Tabel 1 Pengelompokan Algoritma berdasarkan Notasi big Oh

Kompleksitas algoritma tersebut memiliki urutan yang menunjukkan tingkat kompleksitas suatu algoritma sebagai berikut.
$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) <.... < O(n!)$

III. ANALYSIS OF TIME EXECUTION AND VISUALIZATION

A. Merge sort
Merge Sort menonjol di antara metode pengurutan karena pendekatan *divide-and-conquer* dan kinerjanya yang terjamin setiap kali. Merge Sort, yang diciptakan oleh John von Neumann pada tahun 1945,

beroperasi dengan membagi *array* menjadi *subarray* yang lebih kecil, mengurutkan setiap *subarray*, dan kemudian menggabungkannya kembali menjadi array yang terurut.

1. Kompleksitas algoritma merge sort

   1.1 Best-Case (Ω(n log n)):

   Best-case terjadi ketika elemen sudah terurut sebelumnya sesuai dengan kunci yang digunakan. Pada Merge Sort, algoritma tetap harus membagi dan menggabungkan meskipun data sudah terurut.

   1.2 Worst-Case (O(n log n)):

   Worst-case terjadi ketika elemen dalam array dalam urutan yang berlawanan atau paling tidak terurut. Hal ini menyebabkan operasi penggabungan harus dilakukan secara penuh di setiap langkah.

   1.3 Average-Case (Θ(n log n)):

   Average-case mencakup skenario di mana elemen-elemen di array diacak secara acak. Dalam kasus ini, kompleksitas tetap O(n log n) karena proses pembagian dan penggabungan tetap konstan.

2. Perhitungan T(n)

   2.1 Merge Sort Rekursif

   Algoritma membagi array menjadi dua bagian secara rekursif hingga setiap bagian hanya memiliki satu elemen. lalu Proses penggabungan membutuhkan waktu O(n) pada setiap level rekursi. Kompleksitas waktu total T(n) dapat direpresentasikan dengan:

$$T(n) = 2T(n/2) + cn$$

Dengan menyelesaikan menggunakan rekursi menggunakan metode substitusi:

$$T(n) = cn\,log_2 + O(n)$$

jadi kompleksitas nya adalah

$$O\,(n\,log\,n)$$

2.2 Merge Sort Iteratif

Algoritma menggabungkan subarray kecil menjadi subarray yang lebih besar dalam iterasi bertahap, lalu Pada setiap langkah iterasi, ukuran subarray yang digabungkan bertambah dua kali lipat.

Kompleksitas waktu total:

$$O\,(n\,log\,n)$$

IV. ANALYSIS OF TIME EXECUTION AND VISUALIZATION

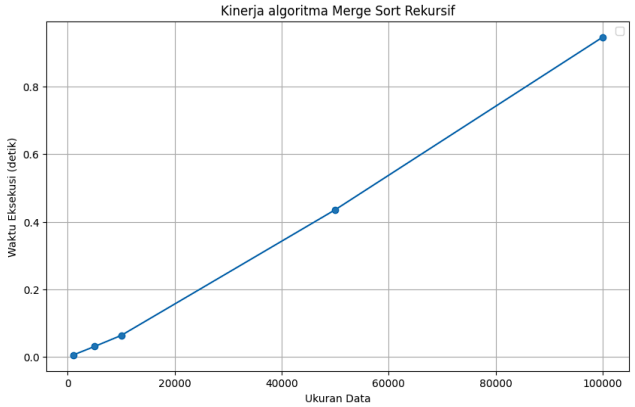
Kami disini menggunakan bahasa pemrograman python dan menggunakan google collab untuk menjalankan program nya

```python
def merge_sort_recursive(arr, left, right, key='price'):
    """Recursive Merge Sort"""
    if left < right:
        mid = (left + right) // 2

        # Rekursif di sisi kiri
        merge_sort_recursive(arr, left, mid, key)

        # Rekursif di sisi kanan
        merge_sort_recursive(arr, mid+1, right, key)

        # Menggabungkan kedua bagian
        merge_recursive(arr, left, mid, right, key)
```

gambar 1 algoritma merge sort recursive

gambar di atas ini adalah potongan kode dari program kami yaitu *merge*

sort recursive dengan melakukan analisis code menggunakan plot yang ada di library python kita mendapatkan graphic seperti ini



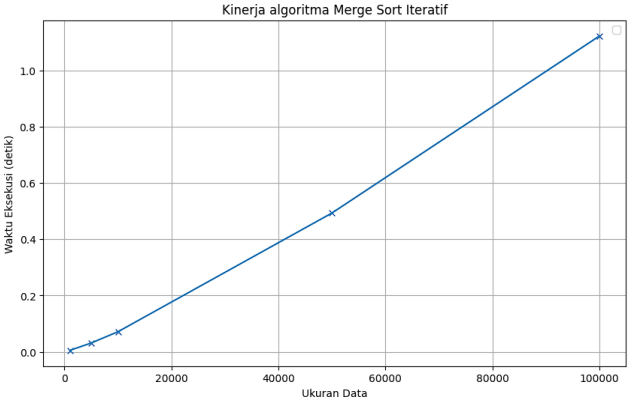
gambar 2 kinerja algoritma sort rekursif

Selanjutnya adalah algoritma *merge sort iterative,* berikut adalah potongan code algoritma merge sort iteratif kami

```python
def merge_sort_iterative(arr, key='price'):
    """Iterative Merge Sort"""
    n = len(arr)
    # Iterasi dengan ukuran subarray yang bertambah
    size = 1
    while size < n:
        # Iterasi untuk menggabungkan subarray
        for left_start in range(0, n, 2*size):
            # Tentukan batas-batas subarray
            mid = min(left_start + size - 1, n-1)
            right_end = min(left_start + 2*size - 1, n-1)

            # Gabungkan subarray
            merge_recursive(arr, left_start, mid, right_end, key)

        # Naikkan ukuran subarray
        size *= 2
```

gambar 3 algoritma merge sort iterative

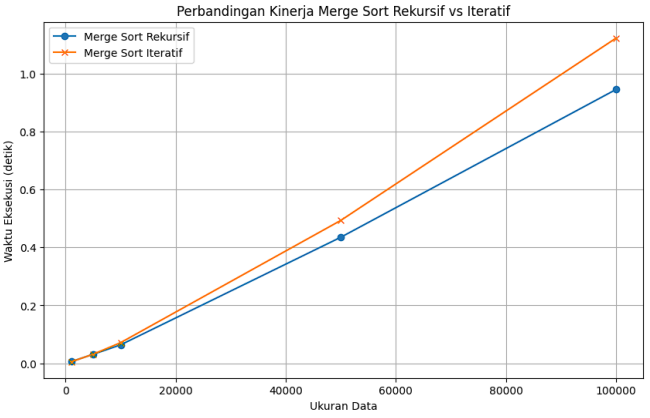
sama seperti sebelumnya kami menggunakan google collab dan melakukan analisis code menggunakan plot yang ada di library python dan mendapat kan graphic seperti ini



gambar 3 kinerja algoritma sort iterative

- merge sort rekursif vs merge sort iteratif

Walaupun memiliki kompleksitas yang sama yaitu $O(n\ log\ n)$ kedua algoritma tersebut memiliki perbedaan bisa di lihat dari graphic berikut ini



gambar 4 perbandingan kinerja iteratif dan rekursif

Kami juga membuat algoritma untuk menghitung hasil perbandingan dari kedua algoritma tersebut dan kami mendapatkan hasil

Ukuran data	Rekursif(detik)	iteratif(detik)
1000	0.0055	0.0048
5000	0.0307	0.0311
10000	0.0635	0.0711
50000	0.4353	0.4934
100000	0.9458	1.1231

Tabel 2 perbandingan rekursif dan iterative

V. CONCLUSION

Dari analisis perbandingan antara algoritma Merge Sort Rekursif dan Iteratif menunjukkan bahwa keduanya memiliki kompleksitas waktu yang sama, yaitu O($n \log n$). Namun, terdapat perbedaan signifikan pada implementasinya. Merge Sort Rekursif cenderung lebih sederhana untuk diimplementasikan karena mengikuti pendekatan top-down dengan pembagian masalah menjadi submasalah yang lebih kecil hingga mencapai ukuran dasar. Di sisi lain, Merge Sort Iteratif menggunakan pendekatan bottom-up, yang menggabungkan subarray kecil terlebih dahulu hingga seluruh array tersortir.

Hasil pengukuran waktu eksekusi menunjukkan bahwa Merge Sort Rekursif sedikit lebih cepat pada ukuran data yang besar dibandingkan dengan versi Iteratif. Ini mungkin karena overhead kontrol loop pada pendekatan Iteratif. Namun, Merge Sort Rekursif membutuhkan lebih banyak memori untuk tumpukan rekursi, yang dapat menjadi masalah bagi sistem dengan memori terbatas.

Secara umum, Merge Sort Iteratif mungkin lebih cocok untuk sistem dengan keterbatasan memori, tetapi Merge Sort Rekursif lebih baik untuk aplikasi yang membutuhkan kode yang lebih sederhana.

Catatan tambahan

sesi ini hanya tambahan saja, kami sudah menjalankan program ini beberapa kali terkadang kami mendapatkan hasil yang berbeda beda tergantung kecepatan laptop dan server dari google collab nya tetapi hasil yang sering muncul adalah apa yang sudah dibahas di makalah ini

REFERENCE

Kumparan, (2023). Contoh Merge Sort, Pengertian, beserta Cara Kerjanya. Diakses dari [https://kumparan.com/how-to-tekno/contoh-merge-sort-pengertian-beserta-cara-kerjanya-20Qu12Ui8J7]

DataCamp (2018), Analyzing Complexity of Code through Python, Diakses dari [https://www.datacamp.com/tutorial/analyzing-complexity-code-python]

Geeksforgeeks (2023), Difference between Recursion and Iteration, Diakses dari [https://www.geeksforgeeks.org/difference-between-recursion-and-iteration/]