Analisis Kebutuhan Waktu Algoritma *Insertion Sort, Merge Sort,* dan *Quick Sort* dengan Kompleksitas Waktu

M. Allbar Pratama
Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sriwijaya

Anita Desiani
Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sriwijaya
anita_desiani@unsri.ac.id

Irmeilyana
Fakultas Matematika dan Ilmu Pengetahuan Alam, Universitas Sriwijaya

***Abstract*---Sorting is a crucial problem in data processing or database. Data processing will be more simple if the data has been sorted. Sorting problem requires special techniques to make the process of sorting faster. The techniques are named as sorting algorithms. The reliability of an algorithm can be measured by its time complexities. The time complexity T(n) is the number of operations performed in an algorithm for N data input. One of time complexities is Big-O or worst case. The Worst case (Big-O) is a time complexities for the worst condition of an algorithm. This study will analyze the time complexity of the algorithms Insertion Sort, Merge Sort and Insertion Sort based on their Big-O (worst case). Each algorithm will be calculated its complexity time in two ways. The first is calculated based on their steps in**

sorting process and the second is calculated based on their coding and running program using C++. The time complexity of Merge Sort is O(n log n) and time complexity of Quick Sort and Insertion Sort is $O(n^2)$, it means the time complexity of Merge Sort is less and faster for large N data input than Quick Sort and Insertion Sort. Otherwise Insertion Sort is faster for small N data input than Merge Sort and Quick Sort. Quick sort needs much time to sort data not only for small N data input but also for large N data input. It means Quick Sort doesn't work well in worst case.*

Keyword:Sorting, Insertion Sort, Quick Sort, Merge Sort, Time Complexity, Worst Case, Big-O

I. Pendahuluan

Pengurutan (*sorting*) merupakan salah satu algoritma yang sangat penting dalam dunia teknologi informasi terutama pada bagian pengolahan data ataupun basis data [1]. *Sorting* (pengurutan) adalah mengurutkan daftar-daftar tertentu dari urutan yang diberikan dimana unsur-unsur yang diurut tersusun secara menaik atau secara menurun[2][3]–[5]. Menurut [6] algoritma pengurutan yang populer antara lain: *Bubble Sort, Selection Sort, Insertion Sort, Heap Sort, Shell Sort, Qucik Sort, Merge Sort, Radix Sort,* dan *Tree Sort.*

Data yang sudah terurut memiliki beberapa keuntungan. Selain mempercepat waktu pencarian, dari data yang terurut dapat langsung diperoleh nilai maksimum dan nilai minimum. Misalnya untuk data numerik yang terurut menurun, nilai maksimum adalah elemen pertama *array*, dan nilai minimum adalah elemen terakhir *array* [1][7].

Banyaknya algoritma pengurutan menunjukkan bahwa persoalan pengurutan adalah persoalan yang kaya dengan solusi algoritmik[8]. Sehingga seorang *programmer* harus menentukan algoritma pengurutan manakah yang membutuhkan kebutuhan waktu algoritma paling sedikit. Kebutuhan waktu suatu algoritma bergunalam menentukan kinerja suat algoritma, sehingga diperlukan kriteria formal yang digunakan untuk menilai algoritma yang terbaik. Kriteria tersebut disebut dengan kompleksitas waktu[9], [10].

Kompleksitas waktu terdiri dari *best case, worst case,* dan *average case* merupakan hal penting untuk mengukur efisiensi suatu algoritma. Kompleksitas waktu dari suatu algoritma yang terukur dianggap sebagai suatu fungsi ukuran masukan [11]. Kompleksitas waktu diekspresikan sebagai jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari ukuran masukan *n*. Dengan menggunakan kompleksitas waktu, laju peningkatan waktu yang diperlukan algoritma dapat ditentukan sesuai dengan meningkatnya ukuran masukan *n* [12].

Menurut [11] analisis kompleksitas waktu algoritma adalah membandingkan waktu yang dibutuhkan algoritma dalam menyelesaikan perintah. [13] juga menyebutkan dengan menganalisis kompleksitas waktu disimpulkan bahwa banyaknya jumlah data-*n* berpengaruh terhadap kebutuhan waktu yang diperlukan. Pada penelitian Sonita dan Nurtaneo [14] dari hasil analisis perbandingan Algoritma *Bubble Sort, Quick Sort*, dan *Merge Sort* disimpulkan bahwa Algoritma *Quick Sort* memiliki kompleksitas waktu yang lebih cepat daripada *Bubble Sort* membutuhkan

ISBN : 979-587-705-4
Vol. 3 No. 1

waktu yang paling lama untuk rata-rata waktu yang dibutuhkan (*average case*).

Pada penelitian ini, algoritma *Insertion Sort*, *Merge Sort*, dan *Quick Sort Sort* diteliti kompleksitas waktunya untuk mengetahui algoritma mana yang paling efisien dari ketiga algoritma tersebut berdasarkan *worst case*-nya.

II. Metode Penelitian

Langkah-langkah yang dilakukan dalam menyelesaikan penelitian ini adalah:

1. Membuat *Pseudo-code* dari Algoritma *Insertion Sort, Merge Sort,* dan *Quick Sort.*
2. Membuat program dari ketiga algoritma tersebut dengan bahasa pemrograman C++.
3. Menganalisis kompleksitas waktu asimptotik *T*(*n*) dari setiap algoritma (*Insertion Sort, Merge Sort,* dan *Quick Sort*).
   3.1. Menghitung banyaknya operasi yang dilakukan algoritma sampai diperoleh *T*(*n*) seperti pada Persamaan (2.2).
   3.2. Mendapatkan Notasi *O*-Besar (*worst case*) *O*(*f*(*n*)) dari hasil Langkah 3.1.
   3.3. Jika *T*(*n*) yang memenuhi Persamaan (2.8) maka Notasi *O*-Besar (*worst case*) ditentukan dengan menggunakan *Master Theorem.*
4. Membandingkan Notasi *O*-Besar dari ketiga algoritma tersebut untuk menentukan algoritma yang paling efisien.
   4.1. Mengelompokkan algoritma berdasarkan kompleksitas waktu asimptotik hasil Langkah 3.
   4.2. Menganalisis urutan spektrum kompleksitas waktu algoritma.
5. Melakukan simulasi dengan menginputkan *n* data pada ketiga algoritma dengan menggunakan program yang dihasilkan dari Langkah 2.
6. Mengkaji dan mengintepretasikan hasil dari ketiga kompleksitas waktu tiap algoritma pada Langkah 5.

III. Hasil dan Pembahasan

Menurut Sedgewick [15] untuk alasan praktis, cukup menghitung jumlah operasi abstrak yang mendasari suatu algoritma dan memisahkan analisisnya dari implementasi.

Jika $T(n) = a_m n^m + a_{m-1} n^{m-1} + \ldots + a_1 n + a_0$ adalah polinomial tingkat *m*, maka $T(n) = O(n^m)$ [16].

Misalkan:

$T_1(n) = O(f(n))$ dan $T_2(n) = O(g(n))$, maka:

(a) $(i)\ T_1(n) + T_2(n) = O(\max(f(n), g(n))$

$(ii)\ T_1(n) + T_2(n) = O(f(n) + g(n))$

(b) $T_1(n)T_2(n) = O(f(n))O(g(n)) = O(f(n)g(n))$

(c) $O(cf(n)) = O(f(n))$, c adalah konstanta

(d) $f(n) = O(f(n))$

1. Insertion Sort

Pada Tabel 1 berikut diperlihatkan operasi yang dilakukan Algoritma *Insertion Sort.*

Tabel 1 Total Perintah Operasi Algoritma *Insertion Sort*

J	Perbandingan	Perpindahan	Total Operasi
2	1	1	2
3	2	2	4
4	3	3	6
5	4	4	8
N	$(n-1)$	$(n-1)$	2(n - 1)

Sehingga total kompleksitas waktu *T*(*n*) untuk *worst case* yang dibutuhkan adalah:

$$T(n) = 2(1) + 2(2) + 2(3) + 2(4) + 2(5) + \cdots + 2(n-1))$$
$$= 2\big(1 + 2 + 3 + 4 + 5 + \cdots + (n-1)\big)$$
$$= 2\frac{(n-1)(n)}{2}$$
$$= (n-1)(n)$$

$T(n) = n^2 - n$ dengan $n > 1$; $n \in \mathbb{Z}$

Ditentukan *worst* case-nya, sehingga:

$T(n)_{Insertion\ Sort} = O(n^2)$

Jadi, *worst case* dari Algoritma *Insertion Sort* adalah $O(n^2)$.

2. Merge Sort

Misalkan *Merge Sort*(*A, p, r*) = *T*(*n*). Untuk perintah operasi *if* ($p < r$) dan $q = L\left[\left(\frac{p+r}{2}\right)\right]$ adalah perintah operasi untuk membagi dua dari *n* data. Sehingga, perintah operasi setelahnya yaitu *merge_sort* (*A, p, q*) dan *merge_sort* (*A, q* + 1, *r*) menerima data sebanyak *n*/2. Kompleksitas waktu dari kedua operasi perintah tersebut adalah *T*(*n*/2) sedangkan kompleksitas waktu dari operasi *merge* adalah *O*(*n*).

Jadi, persamaan kompleksitas waktu untuk *Merge Sort* adalah

$$\left.\begin{aligned} T(n) &= T\left(\tfrac{n}{2}\right) + T\left(\tfrac{n}{2}\right) + O(n) \\ T(n) &= 2\left(T\left(\tfrac{n}{2}\right)\right) + O(n) \end{aligned}\right\} \quad (2.3).$$

dengan $n > 1$; $n \in \mathbb{Z}$

Persamaan diatas adalah persaman rekursif sehingga kompleksitas waktu dapat dicari dengan menggunakan *Master Theorem.*

Diketahui:

$a = 2, b = 2, f(n) = O(n)$, dan $d = 1$ maka

$a = b^d$ yang memenuhi syarat. Sehingga, $T(n) = O(n \log n)$ (4.8)

Jadi, *worst case* dari Algoritma *Merge Sort* adalah $O(n \log n)$.

3. Quick Sort

Karena *Quick Sort* adalah algoritma rekursif maka akan dicari *worst case*-nya dari aturan menghitung Algoritma Rekursif

Kompleksitas waktu $T(n)$ untuk Algoritma *Quick Sort* adalah:

$T(n) = O(n) + T(n-1)$ atau
$T(n) = n + T(n-1)$

Jelas bahwa $T(n) = 1$ untuk $n = 1$ karena hanya mengurutkan satu data. Sehingga didapatkan Persamaan sebagai berikut:

$$T(n) = \begin{cases} 1 \\ n + T(n-1) \end{cases}; \begin{matrix} n = 1 \\ n > 1 \end{matrix}$$

Ambil $T(n) = n + T(n-1)$

$T(n-1) = n - 1 + T(n-2)$

$T(n-2) = n - 2 + T(n-3)$

$T(n) = n + T(n-1)$

$T(n) = n + (n-1) + T(n-2)$

$T(n) = n + (n-1) + (n-2) + T(n-3)$

Sehingga:

$T(n) = n + (n-1) + (n-2) \ldots + (n-k) + T\big(n-(k+1)\big)$

Ambil $T(n-(k+1)), T(n) = 1$ maka:

$n - (k+1) = 1$
$n - k - 1 = 1 \Rightarrow k = n - 2$

sehingga:

$$T(n) = n + (n-1) + (n-2) + \cdots + \big(n-(n-2)\big) + T\big(n-(n-2+1)\big)$$
$$= n + (n-1) + (n-2) + \cdots + (n-n+2) + T(n-n+2-1)$$
$$= n + (n-1) + (n-2) + \cdots + (0+2) + T(0+1)$$
$$= n + (n-1) + (n-2) + \cdots + 2 + T(1)$$

Sehingga:

$$T(n) = n + (n-1) + (n-2) + \cdots + 2 + 1$$
$$= \frac{n(n+1)}{2}$$

$T(n) = \frac{n^2+n}{2}$ dengan $n > 1$; $n \in \mathbb{Z}$

Persamaan (4.19) dapat ditentukan *worst* case-nya dari Teorema 2.1, sehingga:

$T(n)_{Quick\ Sort} = O(n^2)$

Jadi, *worst case* dari Algoritma *Quick Sort* adalah $O(n^2)$.

Tabel 2 Notasi *O*-Besar dari Algoritma *Insertion Sort, Merge Sort,* dan *Quick Sort*

Algoritma	Notasi *O*-Besar (Secara Manual)	Notasi *O*-Besar (Secara *Coding Program*)
Insertion Sort	$O(n^2)$	$O(n^2)$
Merge Sort	$O(n \log n)$	$O(n \log n)$
Quick Sort	$O(n^2)$	$O(n^2)$

Berdasarkan Tabel 2, hasil notasi *O*-Besar secara manual dan secara *coding* program adalah sama. Notasi *O*-Besar untuk algoritma *Insertion Sort* adalah $O(n^2)$, *Merge Sort* adalah $O(n \log n)$, dan *Quick Sort* adalah $O(n^2)$.

4. Pengujian Program *Insertion Sort, Merge Sort*, dan *Quick Sort*.

Pengujian notasi *O*-Besar pada ketiga algoritma tersebut dilakukan dengan cara mengeksekusi *coding* program. Program diuji coba dengan membandingkan berbagai ukuran data yang harus diurutkan. Setelah diuji coba maka didapatkan kebutuhan waktu dari masing-masing algoritma yang disajikan pada Tabel 3.

Tabel 3 Hasil Simulasi Program dengan *Input* Sebanyak *n* Data

n	*Insertion Sort* (Sekon)	*Merge Sort* (Sekon)	*Quick Sort* (Sekon)
10	0.05549	0.06694	0.1477
100	0.07092	0.142	0.08007
1000	0.5879	0.5324	0.6024
10000	7.071	6.713	6.925
45000	41.68	24.99	42

Keterangan: n = banyak data.

Pada Tabel 4.3 dapat dilihat algoritma untuk data yang berukuran $n < 1000$ *Insertion* Sort memerlukan waktu yang lebih cepat dalam men-*sorting* data dibandingkan algoritma *Merge Sort* dan *Quick Sort* pada *worst case*-nya. Untuk data yang berukuran $n \geq 1000$, *Merge Sort* memerlukan waktu yang lebih cepat dalam men-*sorting* data dibandingkan dengan *Insertion Sort* dan *Quick Sort* pada *worst case*-nya.

IV. Kesimpulan

Berdasarkan hasil dan pembahasan dapat disimpulkan bahwa:

(4.13) (4.14) (4.15) (4.16) (4.17) (4.18) (4.19) (4.20)

1. Hasil notasi *O*-Besar tiap algoritma yang dihitung secara manual ataupun secara *coding* program menghasilkan hasil yang sama. Notasi *O*-Besar untuk algoritma *Insertion Sort* adalah $O(n^2)$, *Merge Sort* adalah $O(n \log n)$, dan *Quick Sort* adalah $O(n^2)$.
2. Algoritma *Insertion Sort* membutuhkan waktu yang paling cepat pada masalah *sorting* untuk *input* data *n* yang kecil ($n < 1000$). Untuk data *input n* yang besar ($n \geq 1000$) algoritma *Merge Sort* membutuhkan waktu yang paling cepat sehingga algoritma *Merge Sort* paling efisien dibandingkan dengan algoritma *Insertion Sort* dan *Quick* pada kondisi *worst case*. Algoritma *quick Sort* menghasilkan waktu yang lebih lama untuk menyelesaikan pengurutan baik untuk data *n* yang kecil maupun data *n* yang besar dibandingkan *Insertion Sort* dan *Merge Sort*. Ini berarti *Quick Sort* tidak bekerja baik pada kondisi *worst case*.

Referensi

[1] R. Munir, *Algoritma & Pemrograman Dalam Bahasa Pascal dan C*. Bandung: Informatika, 2011.

[2] S. Palui, S., Gupta, "Unique Sorting Algorithm with Linear Time & Space Complexity," *Int. J. Res. Eng. Technol.*, vol. 3, No. 11, pp. 264–268, 2014.

[3] D. Aho. Alfred, V. Sethi. Ravi. Ullman. Jeffrey, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[4] H. A. Fatta, *Dasar Pemrograman C++ Disertai dengan Pengenalan Pemrograman Berorientasi Objek*. Yogyakarta: Andi, 2006.

[5] B. Raharjo, *Teknik Pemprograman Pascal*. Bandung: Informatika, 2005.

[6] M. Canaan, C. Garai, M. S., Daya, "Popular Sorting Algorithms," *World Appl. Program.*, vol. 1, pp. 62–71, 2011.

[7] S. Mehta, D. P., Sahni, *Handbook of Data Structures and Applications*. USA: Chapman & Hall/CRC Computer and Information Science Series, 2005.

[8] A. Levitin, *Introduction to The Design and Analysis of Algorithms*. Addison-Wesley, 2007.

[9] U. . Azizah, "Perbandingan Detektor Tepi Prewit dan Detektor Tepi Laplacian Berdasarkan Kompleksitas Waktu dan Citra Hasil," *J. UPI.*, vol. 5, 2013.

[10] M. Azmoodeh, *Abstract Data Types and Algorithms*. Macmillan Education, 1988.

[11] D. W. Nugraha, "Penerapan Kompleksitas Waktu Algoritma Prim untuk Menghitung Kemampuan Komputer Dalam Melaksanakan Perintah," *J. Ilm. Foristek*, vol. 2, No. 2, pp. 195–202, 2012.

[12] R. Munir, *Matematika Diskrit*. Bandung: Informatika, 2005.

[13] A. . Estrada S., "Telaah Waktu Eksekusi Program Terhadap Kompleksitas Waktu Algoritma Brute Force dan Divide and Conquer dalam Penyelesaian Operasi List," *J. Ilmu Komput. dan Teknol. Inf.*, vol. 3 No. 2, 2003.

[14] F. Sonita, A., Nurtaneo, "Analisis Perbandingan Algoritma Bubble Sort, Merge Sort, dan Quick Sort dalam Proses Pengurutan Kombinasi dan Huruf," *J. Pseudocode*, vol. 2, no. 2, pp. 75–80, 2015.

[15] R. Sedgewick, *Algorithms in C++*. Addison-Wesley, 1992.

[16] Y. D. Astuti, *Logika dan Algoritma*. Bandung: Universitas Gunadarma, 2005.