Penerapan Konsep Kompleksitas Algoritma terhadap Kompleksitas Waktu Algoritma *Insertion Sort*, *Merge Sort*, dan *Quick Sort*

Laila Bilbina Khoiru Nisa - 13521016
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13521016@std.stei.itb.ac.id

Abstract*—Sorting merupakan masalah krusial dalam pengolahan data atau basis data. Pengolahan data akan lebih sederhana jika datanya telah diurutkan. Masalah penyortiran membutuhkan teknik khusus untuk mempercepat proses sorting. Keefektifan suatu algoritma dapat diukur dari kompleksitas waktunya. Kompleksitas waktu T(n) adalah jumlah operasi yang dilakukan dalam algoritma untuk input data N. Salah satu jenis waktu kompleksitasnya adalah Big-O atau kasus terburuk. Kasus Terburuk (Big-O) adalah kompleksitas waktu untuk kondisi terburuk suatu algoritma. Penulis akan mencoba menganalisis kompleksitas waktu dari algoritma Insertion Sort, Merge Sort dan Insertion Sort berdasarkan Big-O mereka (kasus terburuk). Setiap algoritma akan menghitung waktu kompleksitasnya dengan dua cara. Yang pertama adalah dihitung berdasarkan langkah mereka masuk proses penyortiran dan yang kedua dihitung berdasarkan coding dan menjalankan program menggunakan C++. Waktu kompleksitas Merge Sort adalah O(n log n) dan kompleksitas waktu dari Quick Sort dan Insertion Sort adalah O(n2), artinya waktu kompleksitas Merge Sort lebih sedikit dan lebih cepat untuk data N besar masukan dari Quick Sort dan Insertion Sort. Dan Insertion Sort lebih cepat untuk input data N kecil daripada Merge Sort dan Quick Sort. Quick Sort membutuhkan banyak waktu untuk mengurutkan data tidak hanya untuk input data N kecil tetapi juga untuk input data N besar. Ini berarti Quick Sort tidak berfungsi dengan baik dalam kasus terburuk.

***Keywords*—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 . 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 berguna dalam menentukan kinerja suatu 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.

Pada makalah 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 casenya.

II. DASAR TEORI

Pada bab ini, akan diberikan dasar teori untuk menganalisis kompleksitas waktu dari algoritma yangdisebutkan.

2.1 Algoritma

Algoritma adalah serangkaian atau alur langkah-langkah dalam mengatasi suatu masalah dan mengetahui apa masalahnya, apakah masalah itu logis atau sistematis, menurut pengertian ini. Jika kita perhatikan lebih dekat, kita mungkin

telah mengikuti aturan algoritma dalam kehidupan kita sehari-hari.

Algoritma berasal dari kata algoritmik dan ritmik, yang pertama kali diungkapkan dalam kitab Al-Jabr Wa-al Muqobla karya Abu Ja'far Mohammad Ibn Musa Al Khowarizmi (825M). Algoritma dalam pemrograman mengacu pada strategi tertentu yang dapat diterima dan terdiri dari serangkaian prosedur metodis terstruktur dan tertulis yang harus diikuti untuk menyelesaikan masalah menggunakan komputer.

Dan algoritmanya juga tidak harus mengikuti langkah-langkah standar seperti perhitungan matematis. Algoritma mengajarkan bagaimana kita dapat atau lebih mudah menyelesaikan masalah dengan berbagai solusi dan memilih solusi mana yang terbaik. Apakah solusi A atau solusi B lebih baik, tergantung pada solusi mana yang kita gunakan untuk menyelesaikan masalah tersebut. Ciri-ciri algoritma yang baik adalah:

- Ketika datang untuk memecahkan kesulitan, algoritma memiliki logika atau prosedur penghitungan yang tepat.
- Menghasilkan hasil yang tepat dan akurat dalam waktu singkat.
- Untuk menghindari ambiguitas, metode ini ditulis dalam bahasa standar secara sistematis dan bersih.
- Algoritma dinyatakan dalam format yang sederhana dan mudah dipahami yang dapat dengan mudah diimplementasikan ke dalam bahasa komputer.
- Semua operasi yang diperlukan ditentukan secara rinci.
- Setelah sejumlah langkah tertentu, semua proses dalam algoritma harus sampai pada suatu kesimpulan.

Algoritma pemrograman, di sisi lain, adalah seperangkat prosedur untuk memecahkan masalah pemrograman dengan komputer. Algoritma harus ditulis dalam urutan tertentu agar komputer dapat memahami dan menjalankannya. Untuk membuat suatu algoritma diperlukan analisis kasus, misalnya untuk menentukan proses apa yang diperlukan untuk memecahkan masalah yang harus dipecahkan.

2.2 Perbandingan Nilai (Sorting)

Yahya (2014:135) Sorting adalah proses menempatkan data ke dalam urutan dan urutan berdasarkan aturan eksklusif setelah sebelumnya telah diurutkan secara acak atau tidak teratur. Secara umum, ada dua jenis pengurutan: ascending (dari karakter/angka kecil ke karakter/angka besar) dan descending (dari karakter/angka besar ke karakter/angka kecil). Pengurutan data (sorting) adalah proses menyusun data yang ditempatkan secara acak dengan cara tertentu ke dalam pola teratur yang mengikuti aturan-aturan tertentu. Penyortir ini bekerja baik naik maupun turun, dan dapat mengurutkan data jenis apa pun, termasuk angka dan karakter.

2.3 Kompleksitas Waktu

berdasarkan Traju (2010:10) Ketika T(n) adalah jumlah operasi yang dilakukan untuk menjalankan algoritma sebagai fungsi dari ukuran input, kompleksitasnya adalah T(n). Akibatnya, saat menentukan jumlah operasi yang dijalankan oleh algoritme, kompleksitasnya diukur. Operasi inti dari algoritma pengurutan, terutama pengurutan yang menggunakan perbandingan, adalah operasi perbandingan elemen larik dan operasi pertukaran elemen. Karena jumlah keduanya tidak sama, kedua item tersebut dihitung satu per satu. Secara umum, notasi big-0 digunakan untuk menunjukkan kompleksitas algoritma secara asimtotik. T(n) adalah kompleksitas suatu algoritma saat dijalankan, dan memenuhi

$$T(n) \leq c\left(f\ (n)\right)$$

T(n) = O(f) dapat digunakan untuk menyatakan kompleksitas untuk n n0 (n).

2.4 Quick Sort

C.A.R. Hoare pertama kali mengusulkan Quicksort pada tahun 1960, dan pertama kali diterbitkan sebagai artikel di Computer Journal lima pada bulan April 1962. Ini adalah algoritma pengurutan yang paling banyak digunakan, terutama dalam bahasa pemrograman, karena metode aplikasinya yang mudah, efisien, dan efektif. Sampai saat ini, sejumlah besar penelitian dan pengembangan telah dilakukan. Peneliti seperti Sedgewick, Bentley, McIlroy, Clement, Flajolet, Vallee, dan Martinez mengerjakan analisis dan implementasi quicksort.

Menurut Saputra, dkk (2010:1), Quick Sort adalah algoritma pengurutan berdasarkan contoh Divide and Conquer, dimana output dibagi menjadi dua bagian yang lebih kecil secara bertahap. Quick Sort adalah algoritma pengurutan yang sangat cepat jika dibandingkan dengan algoritma pengurutan lainnya. Quick sort ini melakukan sorting dengan memisahkan masalah menjadi sub-problem, yang selanjutnya Penyortiran lebih cepat meskipun memakan banyak memori karena masalah dipisahkan lebih lanjut menjadi sub-masalah. Tindakan yang harus dilakukan adalah sebagai berikut:

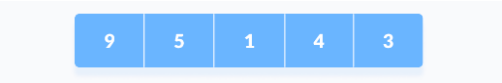
- Dapatkan elemen dari daftar yang dianggap sebagai pivot.
- Susun ulang daftar sehingga elemen dengan nilai lebih kecil dari pivot muncul lebih dulu, diikuti oleh elemen dengan nilai lebih kuat dari pivot (nilai yang sama dapat diputar nanti). Pivot sekarang berada di posisi puncaknya. setelah membelah. Partisi adalah istilah yang digunakan untuk menggambarkan prosedur ini.
- Bagilah potongan-potongan yang lebih kecil menjadi subdivisi lagi daftar diurutkan secara rekursif, dan dari elemen yang lebih besar, sub daftar diurutkan.

Masalah mendasar dengan rekursi adalah bahwa daftar besaran nol atau satu tidak diperlukan untuk pengurutan.

2.5 Insertion Sort

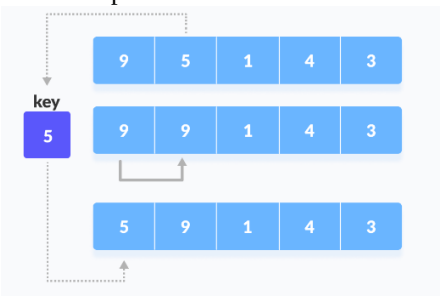
Insertion Sort adalah algoritma pengurutan yang menempatkan elemen yang tidak disortir di tempat yang sesuai di setiap iterasi.

Jenis penyisipan bekerja sama seperti kita mengurutkan kartu di tangan kita dalam permainan kartu. Saya berasumsi bahwa kartu pertama sudah disortir kemudian, dipilih kartu yang tidak disortir. Jika kartu yang tidak disortir lebih besar dari kartu di tangan, maka diletakkan di sebelah kanan sebaliknya, di sebelah kiri. Dengan cara yang sama, kartu lain yang tidak disortir diambil dan diletakkan di tempat yang benar. Pendekatan serupa digunakan oleh insertion sort.



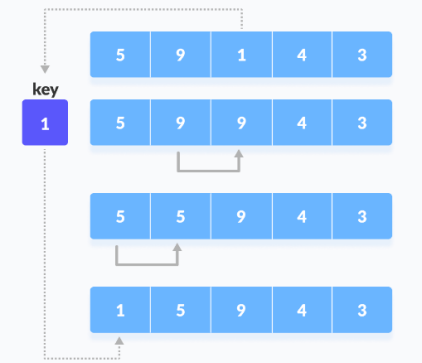
Gambar 1. Sebuah List Array

1. Elemen pertama dalam array diasumsikan diurutkan. Ambil elemen kedua dan simpan secara terpisah di kunci. Bandingkan kunci dengan elemen pertama. Jika elemen pertama lebih besar dari kunci, maka kunci diletakkan di depan elemen pertama.



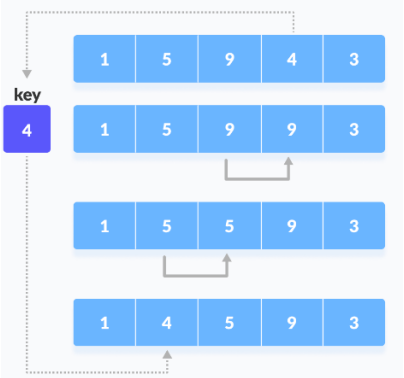
Gambar 2. Ilustrasi dari step 1

2. Sekarang, dua elemen pertama diurutkan. Ambil elemen ketiga dan bandingkan dengan elemen di sebelah kirinya. Menempatkannya tepat di belakang elemen yang lebih kecil darinya. Jika tidak ada elemen yang lebih kecil darinya, maka letakkan di awal array.



Gambar 3. Ilustrasi dari step 2

3. Demikian pula, tempatkan setiap elemen yang tidak disortir pada posisi yang benar.



Gambar 4. Ilustrasi dari step 3

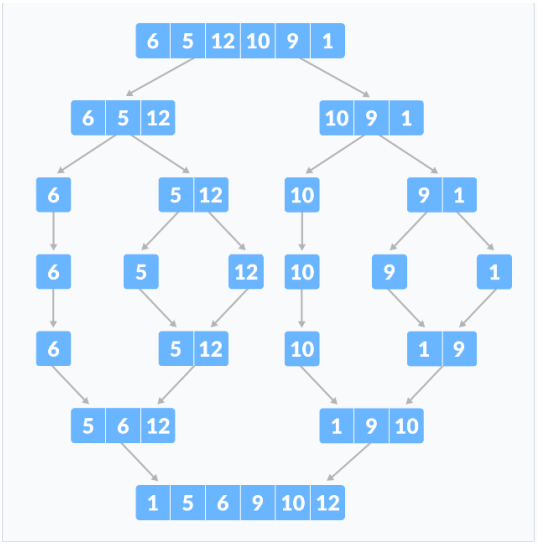
```
insertionSort(array)
  mark first element as sorted
  for each unsorted element X
    'extract' the element X
    for j <- lastSortedIndex down to 0
      if current element j > X
        move sorted element to the right by 1
    break loop and insert X here
end insertionSort
```

Gambar 5. Algoritma Insertion Sort

2.6 Merge Sort

Merge Sort adalah salah satu algoritma sorting paling populer yang didasarkan pada prinsip Divide and Conquer Algorithm.

Di sini, sebuah masalah dibagi menjadi beberapa sub-masalah. Setiap sub-masalah diselesaikan secara individual. Akhirnya, sub-masalah digabungkan untuk membentuk solusi akhir.



Gambar 6. Contoh dari Marge Sort

III. METODE PENELITIAN

Langkah-langkah yang dilakukan dalam menyelesaikan penelitian ini adalah:

- Membuat Pseudo-code dari Algoritma Insertion Sort, Merge Sort, dan Quick Sort.
- Membuat program dari ketiga algoritma tersebut dengan bahasa pemrograman C++.
- 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.
- Membandingkan Notasi O-Besar dari ketiga algoritma tersebut untuk menentukan algoritma yang paling efisien.

- Mengelompokkan algoritma berdasarkan kompleksitas waktu asimptotik hasil Langkah 3.
- Menganalisis urutan spektrum kompleksitas waktu algoritma.

- Melakukan simulasi dengan menginputkan n data pada ketiga algoritma dengan menggunakan program yang dihasilkan dari Langkah 2.
- Mengkaji dan mengintepretasikan hasil dari ketiga kompleksitas waktu tiap algoritma pada Langkah 5.
-

IV. 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 polynomial 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) $f(n) = O(f(n))$

4.1 Insertion Sort

Pada Tabel 1 berikut diperlihatkan operasi yang dilakukan 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)

Tabel 1. Total Perintah Operasi Algoritma Insertion Sort

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 $O(n^2)$.

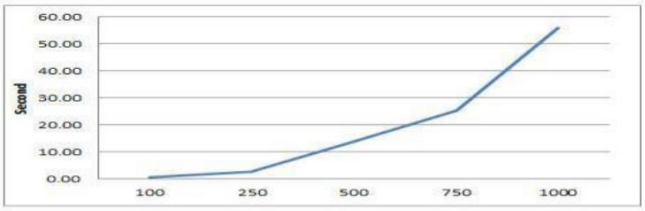
4.1.1 Analisa Insertion Sort

Algoritma ini hanya berjalan satu kali dalam skenario terbaik, yaitu jika item tabel telah diurutkan. Perulangan while tidak digunakan sama sekali. Algoritma ini menjalankan Nmax kali untuk masalah terburuk. Pengurutan berdasarkan penyisipan, seperti pengurutan gelembung, memiliki kompleksitas algoritma $O(n^2)$. .Meskipun memiliki kompleksitas algoritma yang sama, metode pengurutan yang menggunakan penyisipan 2 kali lebih cepat dan lebih efisien daripada pengurutan gelembung ketika dijalankan dengan data input yang sama. Namun, untuk tabel berukuran besar, teknik ini masih kurang efisien (menyimpan nilai poli). Interval data 100 hingga 1000 item digunakan. Detik digunakan untuk mengukur waktu eksekusi (s).

Jumlah elemen *array*	Waktu (s)
100	0.56
250	2.67
500	13.73
750	25.34
1000	56.03

Tabel 2. Waktu eksekusi algoritma Insertion Sort

Grafik berikut mengilustrasikan kompleksitas algoritma Insertion sort:



Gambar 7. Grafik kompleksitas algoritma pengurutan penyisipan

4.2 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:

$$a = 2, b = 2, f(n) = O(n), \text{dan } d = 1$$

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).

4.3 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:

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

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

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

sehingga:

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

Sehingga:

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

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

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

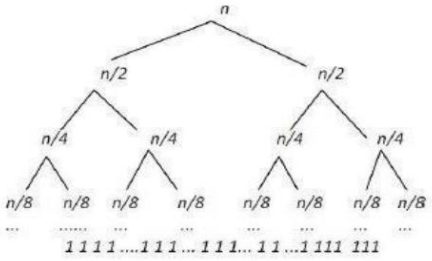
Persamaan (4.19) dapat ditentukan worst case-nya dari Teorema 2.1, sehingga:

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

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

4.3.1 Analisa Quick Sort

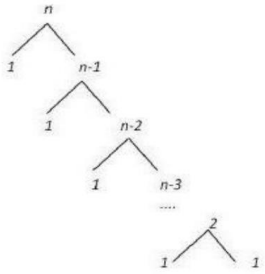
Pemilihan komponen pivot memiliki dampak signifikan pada efisiensi algoritma Quick Sort. Pada setiap level rekursif, pemilihan pivot akan menentukan jumlah dan ukuran partisi. Ketika pivot berada di elemen tengah dan n adalah 2k, dan k konstan, kedua tabel akan selalu berukuran sama di setiap partisi.



Gambar 8. BST dari eksekusi Quick Sort

Di dapatkan kompleksnya adalah : $T(n) = 2.T\left(\frac{n}{2}\right) + c.n \quad , n > 1$

Skenario terburuk adalah ketika komposisi submasalah tidak seimbang (n-1). Ini terjadi ketika pivot pick adalah elemen pertama atau terakhir, menghasilkan partisi pertama 1 dan partisi kedua n-1



Gambar 9. BST scenario terburuk

Metode pengurutan cepat mengurutkan data dengan cepat, namun ini adalah algoritme canggih yang dijalankan secara rekursif. Pendekatan ini sangat cepat untuk beberapa masalah, namun tidak ada yang lebih cepat untuk masalah umum selain algoritma pengurutan cepat. Algoritma pengurutan cepat, di sisi lain, tidak selalu merupakan pilihan terbaik. karena pendekatan ini bersifat rekursif, artinya jika Anda menggunakannya untuk menghasilkan tabel yang sangat besar, akan memakan banyak memori, meskipun cepat.

Pivot awal berada di root node, pivot dari kiri tengah adalah subtree kiri dari root, pivot dari kanan tengah adalah subtree kanan dari root, dan seterusnya untuk setiap eksekusi quicksort menggunakan binary search tree (BST): pivot awal berada di simpul root, pivot dari kanan tengah adalah subpohon kanan dari root, dan seterusnya. Jumlah perbandingan yang dihasilkan dari eksekusi Quicksort sama dengan jumlah perbandingan yang dibuat dengan urutan input selama pembangunan BST. Akibatnya, ketika nilai yang dimasukkan adalah permutasi acak, jumlah perbandingan homogen untuk Quicksort acak sama dengan kasus tipikal untuk membangun BST.

4.4 Perbandingan Kompleksitas waktu

Kompleksitas temporal masing-masing metode memiliki kelebihan dan kekurangannya sendiri jika dibandingkan. Panjang kode, kompleksitasnya, jumlah memori yang digunakan pada saat pemrosesan, kompatibilitas, dan sebagainya. Namun, kompleksitas waktu yang dibutuhkan selalu menjadi masalah utama yang harus diperhitungkan ketika memutuskan algoritma mana yang terbaik dan paling cocok untuk digunakan. Perangkat lunak kemudian menggunakan Algorithm Speed Counter dalam bahasa pemrograman Visual

Basic untuk melacak kecepatan setiap algoritma. Ini digunakan untuk membantu dalam pemeriksaan kecepatan penyortiran data. Atur jumlah n input sinkron aplikasi ke rensom, sehingga tampilan program setiap kali dijalankan berbeda dan tidak dapat diperkirakan, dengan interval data berkisar antara 100 hingga 1000. Pengujian dilakukan menggunakan laptop Asus X200MA dengan Intel® Celeron® CPU N2840 2.16GHz dan RAM 2GB yang menjalankan Windows 8.1 Pro 64-bit.

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	$O(n^2)$	$O(n^2)$ (4.9)

Tabel 3. Notasi *O*-Besar dari Algoritma Insertion Sort, Merge Sort, dan Quick Sort

Berdasarkan Tabel 3, 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 $O(n^2)$.

V. KESIMPULAN

Berdasarkan hasil dan pembahasan dapat disimpulkan bahwa:

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(n2), 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 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.

REFERENCES

[1] R. Munir, Algoritma & Pemrograman Dalam Bahasa Pascal dan C. Bandung: Informatika, 2011. W.-K. Chen, *Linear Networks and Systems* (Book style). Belmont, CA: Wadsworth, 1993, pp. 123–135.

[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.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Desember 2022

Laila Bilbina (13521016)