Jurnal Ilmu Komputer Indonesia(JIK) Vol : 5, No. 2, November 2020
ISSN (Print): 2615-2703, ISSN (Online): 2615-2711

Analisis Komparasi Algoritma *Sorting* Antara Metode *Brute Force* dengan *Divide and Conquer*

[1]I Gede Wira Kusuma Jaya, [2]Ida Bagus Nyoman Wijana Manuaba, [3]Kadek Ryan Wijaya,[4] I Putu Surya Pratama Wardhana, [5]I Made Andika Saputra ,[6] I Gede Aris Gunadi

Program Studi Ilmu Komputer Pasca Sarjana Undiksha

Denpasar

email: kusuma.jaya1@pasca.undiksha.ac.id, wijana.manuaba1@pasca.undiksha.ac.id, ryan.wijaya1@pasca.undiksha.ac.id, pratama.wardhana1@pasca.undiksha.ac.id, andika.saputra1@pasca.undiksha.ac.id

igedearisgunadi@unduiksha,ac.id

Abstrak

Alogritma pengurutan sangat penting dalam pengaplikasian diberbagai bidang praktis Ilmu Komputer. Pengurutan atau *sorting* merupakan suatu proses mengatur susunan data-data sesuai dengan syarat tertentu. Terdapat dua metode pengurutan yang sering digunakan yaitu *brute force* dan *divide and conquer*. Keduanya memiliki cara kerja yang berbeda untuk menyelesaikan masalah. Untuk mengetahui kemangkusan suatu algoritma, perlu dilakukan suatu analisa kompleksitas dalam dimensi waktu dalam mengukur waktu eksekusi dari suatu algoritma dengan notasi *Big O*. Pada makalah ini penulis memberikan analisa terhadap algoritma *sorting* antara metode *brute force* dengan *divide and conquer* pada pengurutan nilai dari sisi kompleksitas dan waktu proses yang jika diterapkan dalam bentuk notasi algoritmik. Dengan melakukan komparasi keduanya, diharapkan dapat memberikan rekomendasi algoritma yang dapat digunakan sebagai pengurutan terbaik.

Kata Kunci: algoritma, pengurutan, brute force, divide and conquer.

Abstract

Sort algorithm is very important in application in various practical fields of Computer Science. Sorting is a process of arranging data arrangement according to certain conditions. There are two ordering methods that are often used, namely brute force and divide and conquer. Both have different ways of working to solve problems. In order to determine the efficiency of an algorithm, it is necessary to carry out a complexity analysis in the time dimension in measuring the execution time of an algorithm with Big O notation.In this paper the author provides an analysis of the sorting algorithm between the brute force method with divide and conquer on the sorting of values in terms of complexity and processing time which when implemented is in the form of algorithmic notation. By comparing the two, it is expected to provide recommendations for the algorithm that can be used as the best sort.

Keywords: algorithm, sorting, brute force, divide and conquer.

I. Pendahuluan

Data adalah kumpulan dari fakta, konsep, atau instruksi pada penyimpanan yang digunakan untuk komunikasi, perbaikan dan diproses secara otomatis yang mempresentasikan informasi yang dapat dimengerti oleh manusia[1]. Data tersimpan secara elektronik dalam sebuah basis data yang terkomputerisasi.

Untuk dapat mengolah data diperlukan suatu algoritma yang dipilih sesuai dengan kebutuhan. Algoritma adalah urutan instruksi yang tidak ambigu untuk memecahkan masalah[2]. Algoritma pengurutan bertujuan mengatur ulang item dari daftar yang diberikan dengan urutan tanpa mengurangi nilai atau isi di dalamnya[2]. Dalam memilih algoritma perlu memperhitungkan efisiensi dengan kompleksitas waktu dan ruang. Tujuan efisiensi adalah untuk menemukan waktu yang paling cepat dalam melakukan proses, sedang tujuan kompleksitas untuk mengetahui seberapa kompleks algortima tersebut bekerja.

Sebagai contoh dalam menilai kemiripan data, kita dapat mengunakan algoritma Rabin-Karp yang menggunakan *fingerprints* untuk menemukan kemunculan satu *string* ke *string* lain, mengurangi waktu perbandingan dua urutan dengan menetapkan nilai *hash* ke setiap *string* dan kata[3]. Algoritma deteksi kemiripan data ini mengeluarkan hasil berupa nilai kemiripan antar data yang tersimpan, kemudian harus diurutkan kembali dari nilai kemiripan terbesar sampai terkecil. Inilah salahsatu fungsi dari pengurutan dalam menyelesaikan permasalahan untuk menilai kemiripan data.

Semakin besar data yang diolah, semakin besar pula waktu yang akan dihabiskan dalam menyelesaikan masalah. Disinilah perlu memilih metode yang tepat khususnya dalam hal pengurutan data. Terdapat dua metode pengurutan yang sering digunakan, taitu *brute force* dan *divide and conquer* yang berguna dalam mengurutkan data dari hasil pencarian tersebut. *Brute force* adalah sebuah pendekatan yang langsung (*straightforward*) untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah (*problem statement*) dan definisi konsep yang dilibatkan, memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas (*obvious way*)[4]. Dalam ilmu komputer, *divide and conquer (D&C)* adalah desain algoritma yang penting dengan basis paradigma multi cabang. Sebuah algoritma divide and conquer bekerja dengan memecahkan masalah secara rekursif kedalam dua atau lebih sub masalah yang sama atau tipe yang berhubungan sampai cukup untuk memecahkan masalah secara langsung. Solusi dari sub masalah lalu di gabungkan untuk memberikan solusi dari masalah yang sebenarnya[5]. Antara metode *brute force* dan *divide and conquer*, memiliki kompleksitas tersendiri dalam menyelesaikan masalah pengurutan data. Dengan melakukan komparasi keduanya, diharapkan dapat memberikan rekomendasi algoritma yang dapat digunakan sebagai pengurutan terbaik.

II. PEMBAHASAN

Algoritma-algoritma pengurutan (*sorting*) dapat diklasifikasi berdasarkan teknik yang digunakan dalam algoritma[6], [7], [8]. Kami mengunakan algoritma *bubble sort* yang diklasifikasikan dalam metode *brute force* sedangkan pada metode *divide and conquer* digunakan algoritma *merge sort*. Dalam melakukan simulasi komparasi antara algoritma *bubble sort* dan *merge sort*, kami menggunakan data hasil penelitian aplikasi deteksi kemiripan karya ilmiah[9] yang berupa nilai desimal persentase kemiripan dokumen yang masih belum terurut.

II. 1 Brute Force

Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas[4]. Namun, algoritma bisa sangat lambat dalam menyelesaikan masalah untuk beberapa kasus.

II.1.1 Konsep Bubble Sort

Bubble sort adalah suatu metode pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya, jika elemen sekarang > elemen berikutnya maka posisinya ditukar, kalau tidak, tidak perlu ditukar[10], [11]. Algoritma *bubble sort* merupakan algortima yang dapat dikatakan paling lama dan prosesnya sangat mudah dilakukan. Tinggal diatur apakah data yang akan dirutkan mulai dari data terkecil sampai terbesar atau dimulai dari data terbesar sampai terkecil.

Proses pengurutan dapat dilakukan dari data awal atau dari data akhir. Jika dimulai dari data awal, maka data paling awal akan dibandingkan dengan data berikutnya, jika data awal lebih besar dari data berikutnya maka tukar posisi (*swap*). Dan pengecekan yang sama dilakukan terhadap data yang selanjutnya sampai dengan data yang terakhir. Kebalikanya, jika dimulai dari data paling akhir maka data paling akhir akan dibandingkan dengan data didepanya, jika data paling akhir lebih kecil dari data didepanya maka tukar posisi.

Proses pada algoritma *bubble sort* dilakukan tahap per tahap. Misal, jumlah data dinyatakan dalam *n* dan jumlah data adalah 8 maka

$$n = 8$$

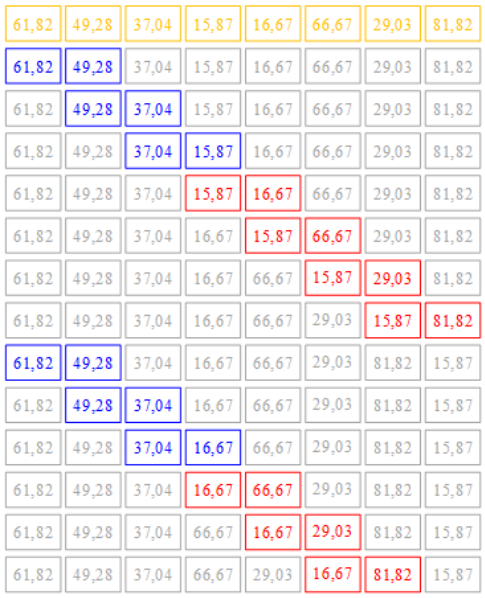
Sehingga akan dilakukan

$$(n-1) = 7\ tahap$$

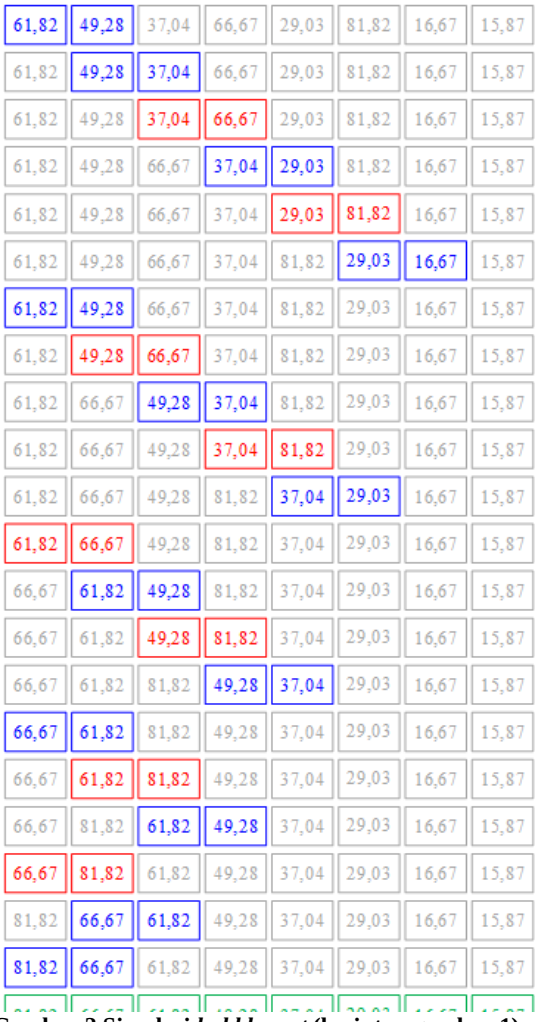
Yang dimulai dari 0 sampai dengan *n-2*.

II.1.2Simulasi Bubble Sort

Sesuai dengan konsep algoritma *bubble sort*, maka simulasi prosesnya ditunjukan pada gambar 1-2 simulasi *bubble sort*.



Gambar 1 Simulasi *bubble sort*



Gambar 2 Simulasi *bubble sort* (lanjutan gambar 1)

Keterangan:

- Kotak kuning menandakan *array* berlum terurut
- Kotak biru menandakan perbandingan nilai *array* sudah terurut
- Kotak merah menandakan perbandingan nilai *array* belum terurut sehingga perlu dilakukan *swap*
- Kotak hijau menandakan *array* sudah terurut

Perbandingan nilai dilakukan antar 2 *array*, jika nilanya tidak sesuai maka akan ditukar posisinya sampai semua *array* terurut dengan benar.

II.1.3 Pseudocode Algoritma Bubble Sort

Algoritma *bubble sort* memiliki bentuk *pseudocode* sebagai berikut[2] dan berubah sedikit dikarenakan data diurutkan dari terbesar sampai terkecil:

```
bubblesort(A[0..n − 1])
for i ← 0 to n − 2 do
  for j ← 0 to n − 2 − i do
    if A[j + 1] > A[j]
      swap A[j] and A[j + 1];
```

1.1.1 Kompleksitas Bubble Sort

Kompleksitas algoritma *bubble sort* terbagi menjadi beberapa kondisi atau kasus, yakni *best case*, *worst case* dan *average case*.

1) *Best Case*
   Kondisi ini terjadi apabila seluruh data sudah dalam keadaan terurut sepenuhnya sehingga dimungkinkan *swap* tidak terjadi.
$$T(n) = n - 1 \approx O(n)$$
2) *Worst Case* dan *Average Case*
   Kondisi ini terjadi apabila data tidak terurut atau acak, baik *worst case* (data terurut terbalik) dan *average case* memiliki nilai kompleksitas yang sama. Hal ini dapat terjadi dan pengulangan sebanyak *n-1* kali, sehingga
$$T(n) = \sum_{i-0}^{n-2} \sum_{j=0}^{n-2-i} 1$$
$$T(n) = \sum_{i=0}^{n-2} [(n-2-i) - 0 + 1]$$
$$T(n) = \sum_{i=0}^{n-2} (n-1-i)$$
$$T(n) = \frac{n(n-1)}{2} \approx O(n^2)$$

Jadi, algoritma *bubble sort* memiliki memiliki 2 nilai kompleksitas untuk kasus berbeda.

II.2 Divide and Conquer

Divide and conquer memecahkan masalah dengan memecahnya menjadi sub-masalah yang lebih kecil, lalu menggabungkan hasilnya untuk menghasilkan solusi atas masalah aslinya[12]. Sekumpulan data angka acak dipecah menjadi bagian-bagin kecil lalu dibanding kemudian diurutkan dan algoritma *merge sort* diklasifikasikan sebagai *divide and conquer*. Algoritma *Merge sort* membagi item yang akan diurutkan menjadi dua bagian, dan secara rekursif mengurutkan masing-masing bagian tersebut, lalu menggabungkannya sampai berakhir[8].

II.2.1 Konsep Merge Sort

Terdapat 3 langkah-langkah utama yang digunakan dalan proses pengurutan dalam *merge sort*, yaitu[13], [14] :

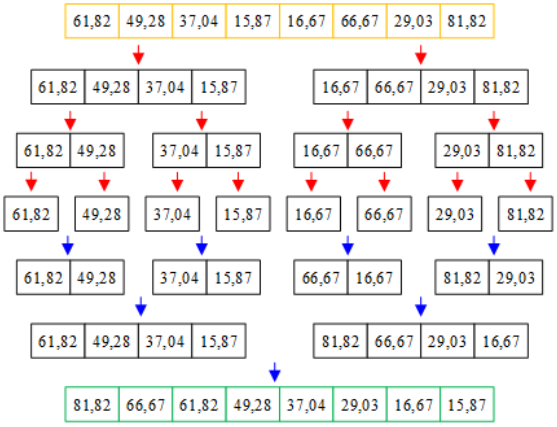
1) *Divide* : Membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil (idealnya berukuran hampir sama). Array A[s...] dipartisi (disusun ulang) menjadi dua (mungkin kosong) sub array A[s...p-1] dan A[p+1...e]. Sub ini array yang dihasilkan dengan memasukkan elemen sedemikian rupa sehingga setiap elemen A[s...p-1] kurang dari atau sama dengan A[p], yang pada gilirannya, kurang dari dan sama dengan setiap elemen A [p+1...e]. Indeks p disesuaikan dengan prosedur partisi.
2) *Conquer*: Memecahkan (menyelesaikan) masing-masing upa-masalah (secara rekursif). Dua sub array A[s...p-1] dan A [p+1...e] diurutkan semacam penyisipan adaptif dalam panggilan rekursif ke prosedur menggunakan buffer tunggal.

3) *Combine*: Menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula. Untuk menggabungkannya tidak dibutuhkan prosedur, keseluruhan array A [s...e] sekarang diurutkan

Selain dengan konsep membagi dan menaklukan, hasil akhir tetap harus digabungkan.

II.2.2 Simulasi Merge Sort

Simulasi algoritma *merge sort* memberikan gambaran bagaiman algoritma tersebut bekerja dalam mengurutkan data, mulai dari data terbesar sampai data terkecil.



Gambar 3 Simulasi *Merge Sort*

Keterangan:

- Tanda panah merah menandakan *array* dipecah (*divide*)
- Tanda panah biru menandakan *array* "dikuasai" diurutkan atau diselesaikan (*conquer*) dan digabung (*combine*)
- Kotak kuning menandakan *array* berlum terurut
- Kotak hijau menandakan *array* sudah terurut

Seperti yang tergambar dalam simulasi *merge sort*, algoritma tersebut memulai prosesnya dari memecah, membandingkan dan akhirnya digabungkan.

II.2.3 Pseudocode Algoritma Merge Sort

Terdapat dua fungsi yang terdapat pada algoritma *merge sort*, yakni *mergesort* dan *merge*[2]. Sama seperti algoritma *bubble sort*, sedikit mengalami perubahan pada bagian membandingkan datanya, karena memerlupakan pengurutan data secara *descending* (data terbesar sampai data terkecil)

```
mergesort(A[0n-1])
if n > 1
copy        A[0..n/2-1]         to
B[0..n/2-1];
copy        A[n/2..n-1]         to
C[0..n/2-1];
mergesort(B[0..n/2-1]);
mergesort(C[0..n/2-1]);
merge(B,C,A);

merge(B[0..p-1],C[0..q-1],A[0..
p+q-1])
i ← 0;
j ← 0;
k ← 0;
while i < p and j < q do
  if B[i] ≥ C[j]
     A[k] ← B[i]; i ← i + 1;
  else
     A[k] ← C[j]; j ← j + 1;
     k ← k + 1;
if i = p
  copy        C[j..q-1]         to
A[k..p+q-1];
else
  copy        B[i..p-1]         to
A[k..p+q-1];
```

II.2.4 Kompleksitas Merge Sort

Kompleksitas dari *merge sort* dapat dicari menggunakan pohon rekursif dan melihat sifat rekursifnya. Jumlah perbandingan yang dilakukan menggunakan merge sort adalah jumlah perbandingan di kiri, jumlah perbandingan di kanan, dan jumlah penggabungan[15].

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

Dengan adanya proses rekrusif pada fungsi *mergersort*, sehingga

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Dengan menelusuri rekurensi yang terjadi ataupun menggunakan pohon rekursif, sehingga

$$T\left(\frac{n}{n}\right) = \frac{2T\left(\frac{n}{2}\right)}{n} + \frac{n}{n}$$

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

$$T\left(\frac{n}{n}\right) = \frac{T\left(\frac{n}{4}\right)}{\frac{n}{4}} + 1 + 1$$

$$\ldots.$$

$$T\left(\frac{n}{n}\right) = \frac{T\left(\frac{n}{n}\right)}{\frac{n}{n}} + 1 + 1 + \cdots + 1 = \log n$$

$$T(n) = n \log n \approx O(n \log n)$$

Kompleksitas algoritma ini berlaku untuk semua kasus, baik *best case*, *worst case*, maupun *average case*[15]. Hal ini dikarenakan baik data sudat terurut maupun tidak terurut, proses yang sama akan tetap berlangsung.

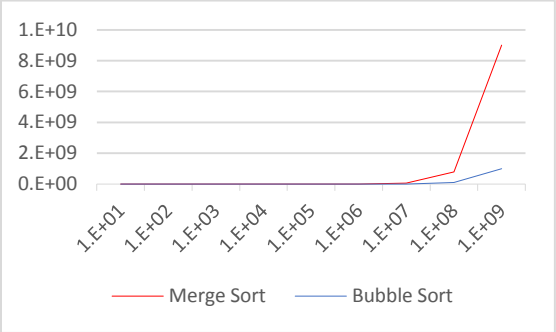
II.3 Komparasi Algoritma

Berdasarkan pembahasan algoritma *bubble sort* dan *merger sort*, didapatkan hasil yang terlihat pada tabel 1. Nilai kompleksitas tersebut sudah sesuai dengan berbagai macam sumber buku[2] maupun jurnal[15]–[17] lainnya.

Tabel 1 Kompleksitas Algoritma

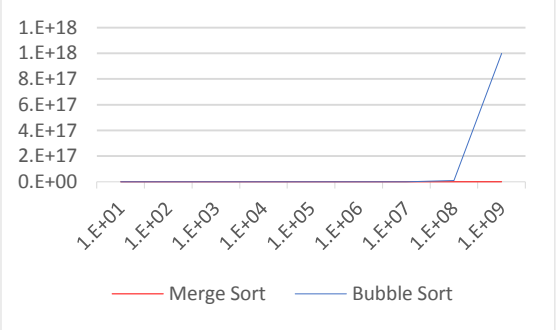
Algoritma	Best Case	Worst Case	Average Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Merge Sort	$n \log n$	$n \log n$	$n \log n$

Kompleksitas dari algortima[17] O(n) bersifat linear, O(n^2) bersifat kuadratik dan n log n berisifat logaritmik, dimana ketiga bentuk tersebut adalah *easy-solved* (*problems*). Jika digambarkan dalam sebuah grafik maka perbandingan antar kompleksitas algoritma akan terlihat pada gambar 4 dan gambar 5.



Gambar 4 Grafik Perbandingan Keadaan *Best Case*

Dalam keadaan *best case*, algoritma *bubble sort* sangat diunggulkan. Namun perlu dingat, bahwa sangat kecil kemungkinan kasus seperti ini terjadi dan tujuan dari pembahasan ini juga adalah menyelesaikan kasus data acak.



Gambar 5 Grafik Perbandingan Keadaan *Worst Case* dan *Average Case*

Keadaan yang paling sering dialami adalah keadaan data yang acak, seperti kasus hasil kemiripan karya ilmiah yang berupa nilai persentase keluaran algoritma Rabin-Karp. Dengan memandang keadaan terburuk (*worst case*) maupun kasus rata-rata kejadian data acak (*average case*), didapatkan perbandingan yang memperlihatkan algoritma *merge sort* lebih baik daripada algoritma *bubble sort*.

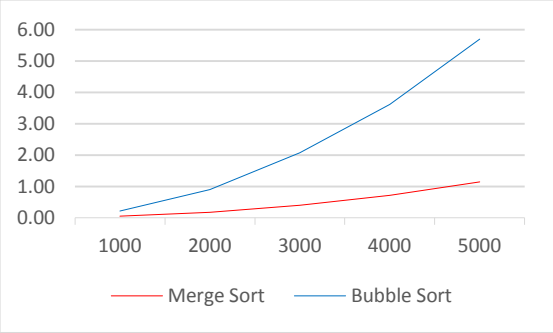
Berdasar pada *pseudocode* masing algoritma, diterapkanlah kedua algoritma tersebut ke dalam program dengan menggubakan bahasa pemrograman PHP[18]. Sedikit mengenai lingkungan jalannya kode tersebut, kode diterapkan berbasis web (PHP 5.6.8) menggunakan server localhost (XAMPP 3.2.1) dan dijalankan pada browser Mozilla Firefox 47.0.2 dengan kemampuan perangkat komputer 2.40Gz.

Skenarionya, program akan membuat data bilangan desimal acak mulai dari nilai 0 sampai dengan 5000 (2 angka penting di belakang koma) yang tersimpan pada sebuah *array*. Data bilangan desimal acak digunakan karena karakteristik data tersebut hampir sama dengan data keluaran perhitungan berbagai macam algortima lain yang ada, termasuk algortima Rabin-Karp.

Proses dijalankanya program dimulai dengan 1.000 data, lalu kemdian bertambah sebanyak 1.000 data di percobaan selanjutnya hingga mencapai 5.000 data uji, jadi terdapat 5 kali percobaan. Masing-masing algoritma dijalankan dan diukur rata-rata waktu prosesenya. Dalam setiap kali percobaan, dilakukan 7 kali penghitungan waktu kemudian dirata-ratakan untuk memperoleh rata-rata waktu prosesnya.

Tabel 2 Rata-rata Waktu Proses Algoritma

Jumlah Data	Bubble Sort	Merge Sort
1.000	0,22 detik	0,05 detik
2.000	0,91 detik	0,18 detik
3.000	2,08 detik	0,41 detik
4.000	3,61 detik	0,72 detik
5.000	5,69 detik	1,15 detik



Gambar 6 Grafik Perbandingan Rata-rata Waktu Proses Algoritma

Dengan melihat rata-rata waktu proses hasil dari percobaan pada tabel 2 dan diperjelas dengan grafik perbandingannya pada gambar 6 semakin memperkuat alasan bahwa algoritma *merge sort* lebih optimal dibandingan dengan algoritma *bubble sort*.

2. KESIMPULAN

Berdasarkan pembahasan algortima *bubble sort* dan *merge sort* yang masing-masing mewakili karakteristik proses *brute force* dan *divide and conquer*, sehingga kami dapat mengambil kesimpulan sebagai berikut

1) Dalam keadaan *best case*, algoritma *bubble sort* memiliki kompleksitas yang lebih baik.
2) Dalam keadaan *worst case* dan *average case*, algoritma *merge sort* memiliki kompleksitas yang lebih baik.
3) Algoritma *merge sort* dapat dikalahkan dengan algoritma dengan metode *brute force* pada keadaan *best case*. Karena algoritma ini akan tetap memecah masalah ke sub-sub masalah yang lebih kecil. Kebalikannya, algoritma ini dapat menyelesaikan masalah lebih cepat pada kondisi worst case daripada *brute force*.
4) Kedua metode ini memiliki kelebihan dan kekurangan masing-masing, oleh sebab itu dalam penerapannya perlu memilih sesuai kondisi/masalah yang ingin diselesaikan.

Daftar Pustaka

[1] W. H. Inmon, *Building the Data Warehouse*, *Fourth Edition*, vol. 13, no. 401. 2005.

[2] A. Levitin, *Introduction to the design and analysis of algorithms*. 2012.

[3] S. Kiran Shivaji and A. Professor, “Plagiarism Detection by using Karp-Rabin and String Matching Algorithm Together Prabhudeva S,” *Int. J. Comput. Appl.*, vol. 116, no. 23, pp. 975–8887, 2015.

[4] B. W. Santoso, F. Sundawa, and M. Azhari, “Implementasi Algoritma Brute Force Sebagai Mesin Pencari (Search Engine) Berbasis Web Pada Database,” *J. Sisfotek Glob.*, vol. 6, no. 1, pp. 1–8, 2016.

[5] M. Z. Karim and N. Akter, “DIVIDE-AND-CONQUER ALGORITHM FOR SOLVING CLOSEST-PAIR PROBLEM,” vol. 3, no. 5, pp. 211–219, 2011.

[6] N. K. K. Govindaraju, N. Raghuvanshi, M. Henson, and D. Manocha, “A cache-efficient sorting algorithm for

database and data mining computations using graphics processors," *Univ. North Carolina, Tech. Rep*, 2005.

[7] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," *IPDPS 2009 - Proc. 2009 IEEE Int. Parallel Distrib. Process. Symp.*, no. May, pp. 1–10, 2009.

[8] Y. Mukhlis and L. Harmanto, "Metode Sorting Bitonic Pada GPU," no. 100, pp. 1–6.

[9] D. A. Putra, H. Sujaini, and H. S. Pratiwi, "Implementasi Algoritma Rabin-Karp untuk Membantu Pendeteksian Plagiat pada Karya Ilmiah," *J. Sist. dan Teknol. Inf.*, vol. 1, no. 1, pp. 1–9, 2015.

[10] D. Kumalasari, "Analisa Perbandingan Kompleksitas Algoritma Bubble Sort , Cocktail Sort Dan Comb Sort Dengan Bahasa Pemrograman C ++," vol. 5, no. 1, pp. 14–20, 2017.

[11] S. Y. Yahya, "Analisa Perbandingan Algoritma Bubble Sort dan Selection Sort Dengan Metode Perbandingan Eksponensial," *Pelita Inform. Budi Darma*, no. April, p. 135, 2014.

[12] R. Rugina and M. Rinard, "Recursion unrolling for divide and conquer programs," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2001, vol. 2017, pp. 34–48.

[13] M. K. Patel and M. S. Yagnik, "A New Approach of Sorting Using Recursive Partition," vol. 3, no. 5, pp. 5–8, 2014.

[14] F. T. Universitas *et al.*, "MAKALAH ALGORITMA DIVIDE AND CONQUER Galih Pranowo," *J. Inform.*, vol. 6, no. Hp 08122723774, p. 432, 2004.

[15] H. L. Hanifah, I. T. Bandung, and J. G. Bandung, "Analisis Kompleksitas Waktu Beberapa Algoritma Sorting," 2014.

[16] A. Kazim, "A Comparative Study of Well Known Sorting Algorithms," *Int. J. Adv. Res. Comput. Sci.*, vol. 8, no. 1, pp. 277–280, 2017.

[17] N. Kumar and R. Singh, "Performance Comparison of Sorting Algorithms On The Basis Of Complexity," *Int. J. Comput. Sci. Inf. Technol. Res.*, vol. 2, no. 3, pp. 394–398, 2014.

[18] M. Doyle, "Beginning PHP 5.3," *Begin. PHP 5.3*, vol. 21, p. 841, 2014.