ILKOM Jurnal Ilmiah
Vol 12, No. 2, Agustus 2020, pp.96-103
E-ISSN: 2548 - 7779

Perbandingan Efisiensi Algoritma *Sorting* dalam Penggunaan *Bandwidth*

Desi Anggreani[a,1,*], Aji Prasetya Wibawa[a,2], Purnawansyah[b,3] dan Herman[b,4]

[a] *Universitas Negeri Malang, Jl. Semarang No.5, Malang 65145, Indonesia*
[b] *Universitas Muslim Indonesia, Jl. Urip Sumoharjo No.5, Makassar 90231, Indonesia*
[1] *desiianggreanii@gmail.com;* [2] *aji.prasetya.ft@um.ac.id;* [3] *purnawansyah@umi.ac.id;* [4] *herman@umi.ac.id*
**corresponding author*

INFORMASI ARTIKEL

Diterima : 16 Maret 2020
Diulas : 19 Juli 2020
Direvisi : 30 Juli 2020
Diterbitkan : 27 Agustus 2020

Kata Kunci:
Insertion Sort
Selection Sort
Merge Sort
Waktu Eksekusi
Penggunan Memori

ABSTRAK

Algoritma yang paling sering digunakan adalah algoritma *sorting*. Telah banyak bermunculan algoritma *sorting* yang dapat digunakan, dalam penelitian ini peneliti mengambil tiga algoritma *sorting* yaitu *Insertion Sort, Selection Sort, dan Merge Sort*. Adapun penelitian ini akan menganalisis perbandingan waktu eksekusi dan pengunaan memori dengan memperhatikan jumlah masukkan data setiap algoritma yang digunakan. Data yang digunakan dalam penelitian ini adalah data penggunaan *bandwidth* jaringan Ukhuwah NET yang terhubung di Fakultas Ilmu Komputer dalam bentuk tipe data double. Setelah melakukan implementasi dan analisis dari sisi waktu eksekusi algoritma *Merge Sort* memiliki waktu eksekusi yang lebih cepat dalam mengurutkan data dengan nilai rata-rata waktu eksekusi sebesar 108.593777 ms pada jumlah data 3000. Sedangkan dalam jumlah data yang sama untuk waktu eksekusi paling lama adalah algoritma *Selection Sort* dengan besar waktu eksekusi 144.498144 ms, adapun dari sisi penggunaan memori dengan jumlah data 3000 Algoritma *Merge Sort* memiliki penggunaan memori yang paling tinggi dibanding kedua algoritma lainnya yaitu sebesar 21.444 MB sedangkan kedua algoritma lainnya secara berturut-turut memiliki penggunaan memori sebesar 20.837 MB dan 20.325 MB.

Keywords:
Insertion Sort
Selection Sort
Merge Sort
Execution Time
Memory Usage

ABSTRACT

The most used algorithm is the sorting algorithm. There have been many popping sorting algorithms that can be used, in this study researchers took three sorting algorithms namely Insertion Sort, Selection Sort, and Merge Sort. As for this study will analyze the comparison of execution time and memory usage by considering the number of enter data of each algorithm used. The data used in this study is ukhuwah NET network bandwidth usage data connected in the Faculty of Computer Science in the form of double data types. After implementing and analyzing in terms of execution time merge sort algorithm has a faster execution time in sorting data with an average execution time value of 108.593777ms on the 3000 data count. While in the same amount of data for the most execution time is the Selection Sort algorithm with a large execution time of 144.498144 ms, in terms of memory usage with the amount of data3000 Merge Sort Algorithm has the highest memory usage compared to the other two algorithms which is 21,444 MB while the other two algorithms have a succession of memory usage of 20,837 MB and 20,325MB.

I. Pendahuluan

Pemrograman merupakan salah satu bidang ilmu komputer. Sebuah program dibangun untuk menyelesaikan masalah tertentu dan mempermudah pengguna komputer. Perkembangan teknologi membawa banyak perkembangan dalam bidang pemrograman. Salah satunya adalah pemanfaatan metode atau algoritma

dalam menyelesaikan sebuah masalah dengan *output* yang lebih akurat[1], [2]. Berbagai algortima bermunculan salah satunya adalah algoritma dalam menyelesaikan masalah dalam hal pengurutan data yang biasa disebut dengan *Sorting*[3]. Fakta bahwa proses pencarian dapat dioptimalkan menjadi sangat cepat, jika disimpan dengan cara berurutan adalah alasan pentingnya sebuah *Sorting*[4], [5]. Alasan lain adalah tingginya kebutuhan untuk melakukan penyortiran yang melekat pada aplikasi[6]. Efesiensi pengurutan diperlukan untuk mengoptimalkan kecepatan pemrosesan. Jika algoritma memiliki efisiensi yang tinggi, maka proses eksekusi akan menggunakan sedikit memori dengan waktu yang lebih cepat serta dapat memproses banyak data[7]. Keperluan efesiensi dari sebuah algoritma *Sorting* juga diperlukan oleh pihak *Programmer*. Hasil perbandingan efesiensi masing-masing algoritma akan dapat dijadikan landasan melakukan pemilihan algoritma yang tepat untuk mengiplementasikan yang sesuai dengan kasus dihadapi oleh masing-masing *Programmer*[8].

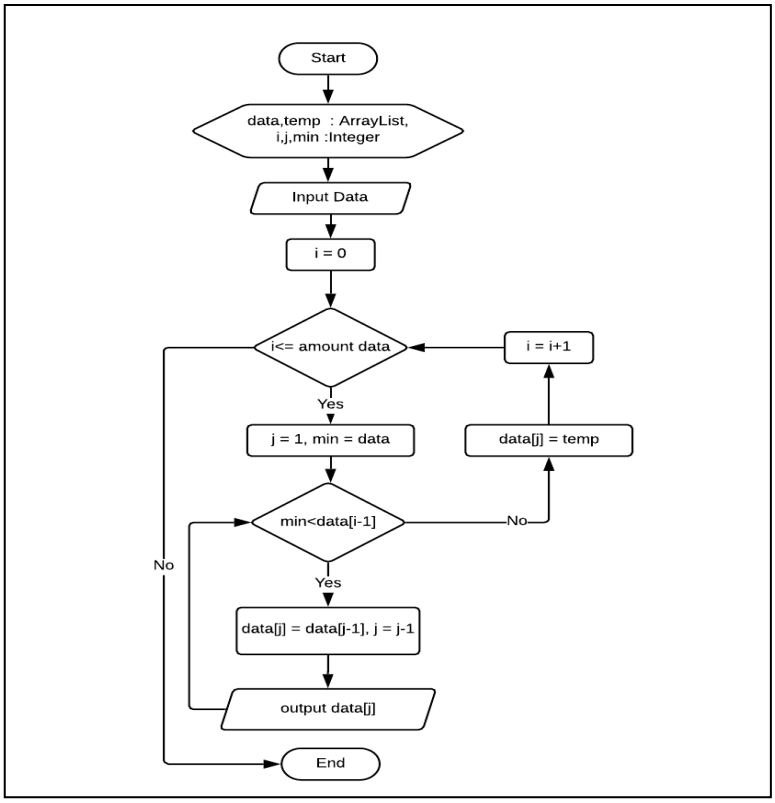
Dari penelitian yang dilakukan oleh Moh Arif Jauhari, dkk (2017) dengan membandingkan waktu eksekusi algoritma *Insertion Sort* dan *Bubble Sort* pada dua Bahasa pemrograman yaitu C# dan GO. Dalam penelitian tersebut pada Bahasa pemrograman C# waktu eksekusi algoritma *Insertion Sort* lebih tinggi dibanding implementasi algoritma *Bubble Sort* pada Bahasa pemrograman GO[9]. Pada penelitian lain juga dilakukan oleh Arief Hendra Saptadi dan Desi Windi Sari (2012) menyatakan bahwa dengan melihat dari sisi waktu eksekusi kinerja algoritma *Merge Sort* memiliki waktu eksekusi yang lebih cepat dibandingkan dengan algoritma *Insertion Sort*[10]. Pada penelitian ini bertujuan melakukan perbandingan efisiensi algoritma *Sorting* khususnya pada *Insertion Sort, Selection Sort, Merge Sort* dilihat dari sisi waktu ekesekusi dan penggunaan memori.

II. Metode Penelitian

Penelitian dilakukan terhadap data set berjumlah 3000 data dengan tipe data yang *double* atau desimal. Dilakukan membagian model data berdasarkan jumalahnya. Data dibagi atas 50, 100, 500, 1000, 3000 data. Dimana data yang diteliti berasal dari penggunaan *Bandwidth* jaringan Ukhuwah Net di Fakultas Ilmu Komputer Universitas Muslim Indonesia yang diambil selama satu pekan. Proses pengurutan data dilakukan dengan 3 algoritma yaitu *Insertion Sort, Selection Sort, Merge Sort*.

A. Insertion Sort

Insertion Sort adalah adalah sebuah algoritma pengurutan yang membandingkan dua elemen data pertama, mengurutkannya, kemudian mengecek elemen data berikutnya satu persatu dan membandingkannya dengan elemen data yang telah diurutkan[11]. Algoritma pengurutan data *insertion sort* akan memeriksa setiap elemen, menemukan yang terkecil atau yang terbesar sesuai kebutuhan jenis pengurutan, dan menyisipkan dalam tempat yang tepat[12]. Adapun *flowchart* Algoritma *Insertion Sort* dapat dilihat pada Gambar 1.



Gambar 1. *Flowchat* Algoritma *Insertion Sort*

Simulasi algoritma *Insertion Sort*:

Data awal:

12	11	96	5	4

Iterasi 1:

11	12	96	5	4

Iterasi 2:

11	12	96	5	4

Iterasi 3:

5	11	12	96	4

Iterasu 4:

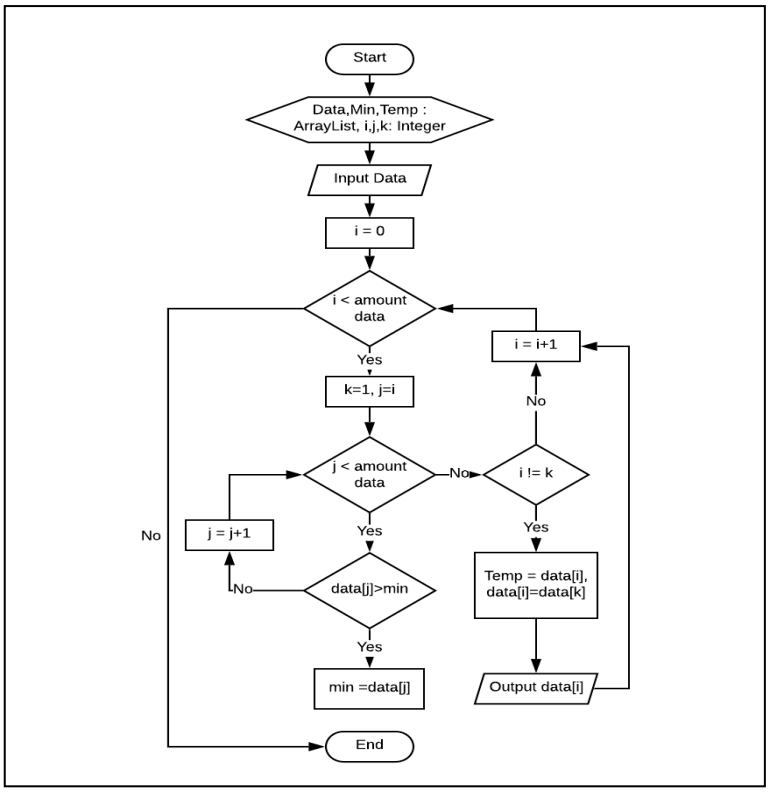
4	5	11	12	96

Kompleksitas waktu eksekusi *Insertion Sort* pada Persamaan (1):

$$T(n) = 1 + 2 + 3 + \cdots + n - 1 = \sum_{f=1}^{n-1} i = \frac{n(n-1)}{2} = O(n^2) \quad (1)$$

B. Selection Sort

Salah satu algoritma penguruatan yang paling sederhana. Algoritma yang lebih dikenal dengan algoritma pemilihan dimana melakukan pemilihan elemen minimum ataupun maksimum. *Selection Sort* adalah suatu Algoritma pengurutan yang membandingkan elemen yang sekarang dengan elemen berikutnya sampai ke elemen yang terakhir. Jika ditemukan elemen lain yang lebih kecil dari elemen sekarang maka dicatat posisisnya dan langsung ditukar. Konsep *Selection Sort* adalah mencari atau memilih nilai terkecil ataupun nilai terbesar dan menukarnya dengan elemen paling awal pada paling kiri pada setiap tahap, proses akan terus berjalan hingga data akan mengahasilkan data yang terurut[13]. Algoritma ini melakukan banyak perbandingan namun memiliki jumlah perpindahan data yang sedikit[14]. Adapun *flowchart* Algoritma *Selection Sort* dapat dilihat pada Gambar 2.



Gambar 2. *Flowchat* Algoritma *Selection Sort*

Simulasi algoritma *Selection Sort*:

Data awal:

12	11	96	5	4

Iterasi 1:

12	11	96	5	4

Iterasi 2:

4	11	96	5	12

Iterasi 3:

4	5	96	11	12

Iterasu 4:

4	5	11	96	12

Iterasu 5:

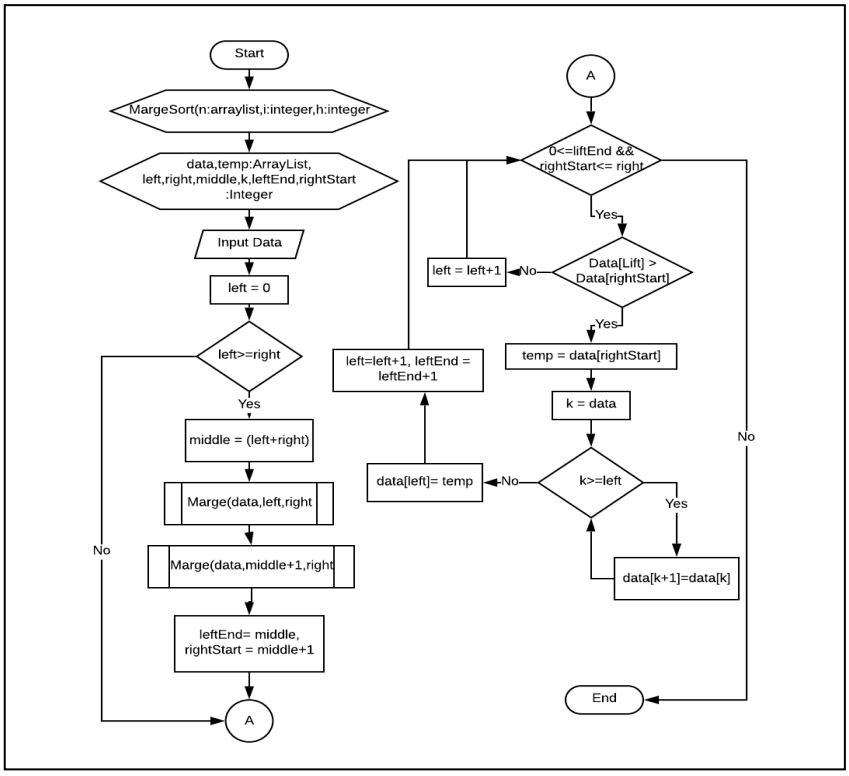
4	5	11	12	96

Kompleksitas waktu eksekusi Selection Sort pada Persamaan (2):

$$T(n) = n - 1 + n - 2 + n - 3 + \cdots + 2 + 1 = \sum_{f=1}^{n-1} i = n - i\ \frac{n(n-1)}{2} = O(n^2) \quad (2)$$

C. Merge Sort

Metode *Merge Sort* menggunakan konsep *devide and conquer* yang membagi data S dalam dua kelompok yaitu S1 dan S2 yang tidak beririsan (*disjoint*). Proses pembagian data dilakukan secara rekursif sampai data tidak dapat dibagi lagi atau dengan kata lain data dalam sub bagian menjadi tunggal. Elemen tunggal akan diurutkan dan terakhir melakukan penggabungan kembali elemen-element yang telah diurutkan menjadi satuan data yang telah terurut dengan memperhatikan keingina pengurutan *ascending* dari kecil ke terbesar atau *descending* dari besar ke kecil [15]. *Merge Sort* akan mendapatkan nilai tengah dengan membagi array menjadi 2 bagian dimana n/2 dan memperoleh sisi kiri dan sisi kanan. Adapun *flowchart* Algoritma *Merge Sort* dapat dilihat pada Gambar 3.



Gambar 3. *Flowchat* Algoritma *Merge Sort*

Simulasi algoritma *Merge Sort*:

Data awal:

12	11	96	5	4

Iterasi 1:

12	11	96	5	4

Iterasi 2:

12	11	96	5	4

Iterasi 3:

11	12	96	4	5

Iterasi 4:

11	12	96	4	5

Iterasi 5:

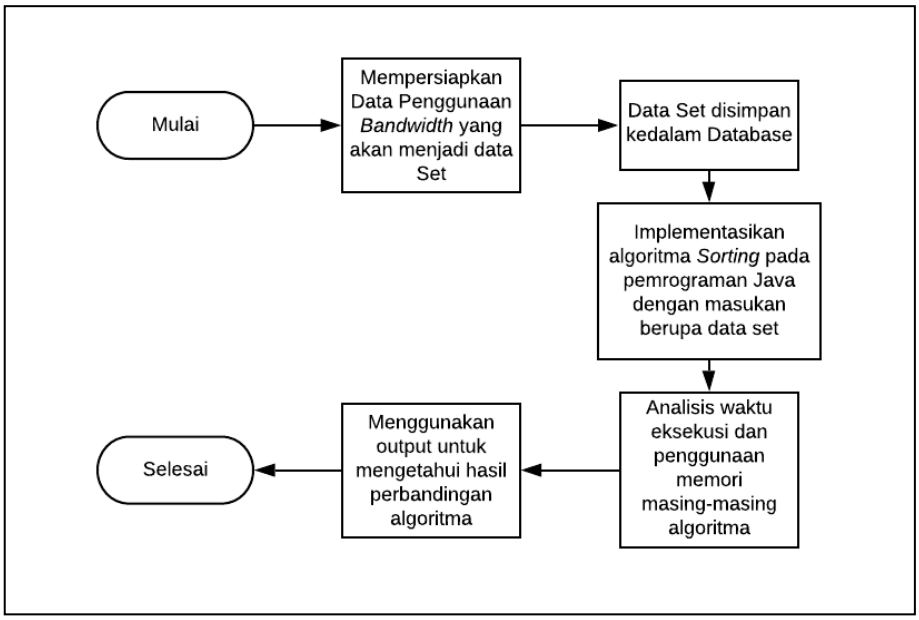
4	5	11	12	96

Kompleksitas waktu eksekusi *Merge Sort*[16] pada Persamaan (3):

$$T(n) = cn \log n + cn = O(n \log n) \quad (3)$$

D. Tahapan Penelitian

Dalam penelitian ini, tahap awal peneliti melakukan pengambilan data *Bandwidth* melalui *tools Winbox-3 RC6* dan memperoleh sebanyak 3000 data. Tahap kedua memasukkan data kedalam database untuk dijadikan Data Set. Data Set yang ada dibagi menjadi beberapa model data berdasarkan jumlah data masing-masing model. Tahap selanjutnya mengimplementasikan ketiga algoritma dalam Bahasa pemrograman java dengan *project* yang berbeda-beda. Setelah melakukan implementasi akan melakukan analisis dari sisi waktu eksekusi dan penggunaan memori masing-masing algoritma berdasarkan hasil penelitian dan teori yang ada. Selanjutnya tahap akhir yaitu melakukan kesimpulan dari analisis yang dilakukan.



Gambar 4. Tahapan Penelitian

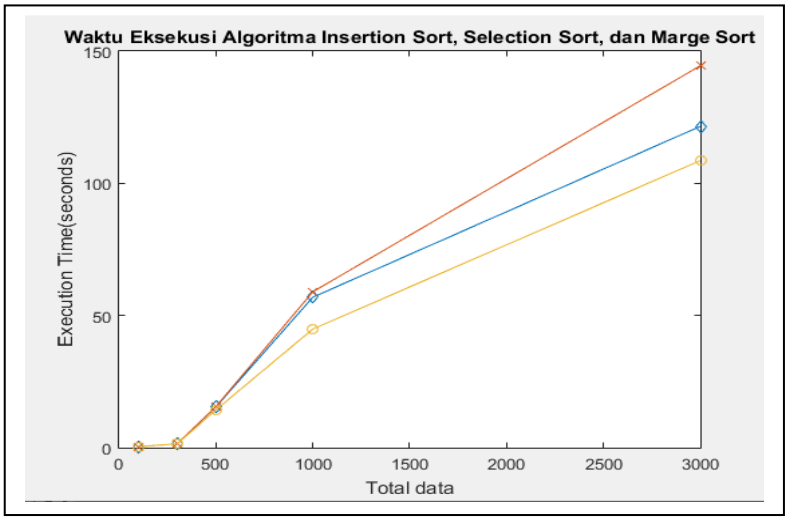
III. Hasil dan Pembahasan

Masing-masing algoritma yang digunakan dalam penelitian ini telah diimplementasikan dengan Bahasa pemrograman Java. Semua algoritma dibuat dengan *project* yang berbeda-beda. Disetiap *project* ada

beberapa *package*, *class* dan *method* yang sama. Selain method untuk Algoritma *sorting*, terdapat beberapa method untuk mengatur parameter dan variabel yang digunakan. Terdapat *method* untuk melakukan *generate* data penggunaan *bandwidth* dari database dan menghitung waktu eksekusi. Waktu eksekusi dihitung menggunakan satuan *miliseconds*. Tahap selanjutnya adalah melakukan analisis dengan hasil dari masing-masing algoritma. Proses pegambilan waktu eksekusi dan penggunaan memori algoritma dilakukan sebayak 10 kali pada tiap algoritma *sorting* dari tiap model data. Selanjutnya, akan dilakukan perhitungan rata-rata dari masing-masing percobaan berdasarkan algoritma *sorting* yang diuji. Hal ini dilakukan untuk melihat konsistensi waktu eksekusi dan penggunaan memori yang dibutuhkan untuk mngurutkan data penggunaan *bandwidth* untuk masing-masing algoritma.

Tabel 1. Rata-rata Waktu Eksekusi Algoritma *Sorting*

Algoritma *Sorting*	**Waktu Eksekusi(*Mili Seconds*)**				
	50 Data	**100 Data**	**500 Data**	**1000 Data**	**3000 Data**
Insertion Sort	0.3546619	1.4655791	15.4466461	57.6217624	121.5711711
Selection Sort	0.394333	1.471437	15.5554208	58.976524	144.498144
Merge Sort	0.392361	1.470433	14.3060158	44.8759382	108.593777

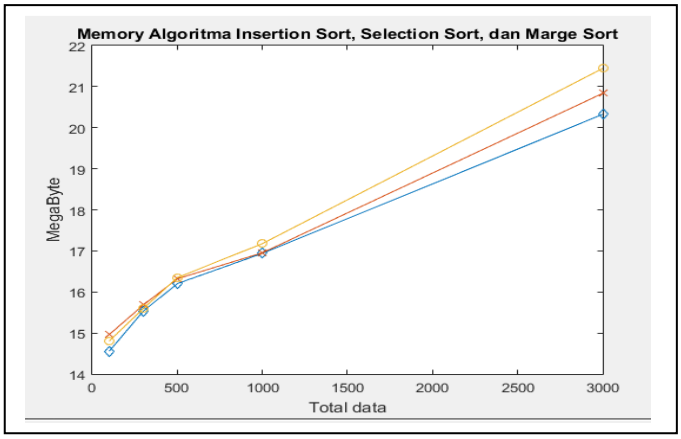


Gambar 5. Grafik rata-rata waktu eksekusi algoritma *sorting*

Berdasarkan Tabel 1 dan Gambar 5 menunjukan jumlah data penggunaan *bandwidth* sebanyak 50 data untuk waktu eksekusi paling cepat adalah algoritma *Insertion sort* dengan waktu 0.3556619 ms dan waktu eksekusi paling lama adalah *Merge sort* dengan waktu 0.394333 ms. Selanjutnya untuk 100 data penggunaan *bandwidth* waktu eksekusi paling cepat adalah masih pada algoritma *Insertion Sort* dengan waktu 1.4655791 ms, sama halnya dengan model 50 data dimana perbedaan waktu eksekusi dari ke tiga algoritma tidak jauh berbeda. Untuk 500 data penggunan *bandwidth* waktu paling tercepat adalah algoritma *Merge sort* dengan waktu 14.3060158 sedikit lebih cepat dibanding *Insertion sort*. Secara konsisten untuk data penggunaan *bandwidth* 1000 dan 3000 algoritma *merge sort* melakukan proses pengurutan paling cepat dibanding dengan dengan algoritma *Insertion sort* dan *Selection sort* dengan masing-masing waktu 44.8759382 dan 108.593777 ms, waktu eksekusi tersebut dengan model data double yaitu data penggunaan *bandwidth.*

Tabel 2. Rata-rata Penggunaan Memori Algoritma *Sorting*

Algoritma *Sorting*	**Penggunaan Memori(*Mega Byte*)**				
	50 Data	**100 Data**	**500 Data**	**1000 Data**	**3000 Data**
Insertion Sort	14.555	15.532	16.203	16.939	20.325
Selection Sort	14.962	15.681	16.320	16.954	20.837
Merge Sort	14.806	15.595	16.349	17.177	21.444



Gambar 6. Grafik rata-rata penggunaan memori algoritma *sorting*

Dari Tabel 2 dan Gambar 6 dilihat penggunaan memori masing-masing algoritma pada model data 50 hingga 500 data perbedaan ketiganya sangan kecil tidak mencapai 1 MB. Berbeda pada model data 1000 dan 3000 algortima *merge sort* secara berturut-turut memiliki waktu eksekusi yang tinggi dibanding algortima *insertion sort* dan *selection sort* yaitu mencapai 17.177 dan 21.444 MB.

IV. Kesimpulan

Secara keseluruhan dalam kasus data penggunaan *bandwidth* yang jumlah data berskala kecil perbedaan dari ketiga algoritma *sorting* masih tergolong sulit untuk ditarik kesimpulan. Namun, pada jumlah data yang berskala besar >1000 penggunaan *bandwidth* berbedaan dari ketiga algoritma sangat terlihat. Dari hasil analisis perbandingan Algoritma *Insertion Sort*, *Selection Sort*, dan *Merge Sort*, dari sisi penggunaan memori pada jumlah penggunaan bandwidth dalam kategori besar algoritma *Marge Sort* memiliki penggunaan memori yang cukup besar namun dari sisi waktu eksekusi pada penggunaan *bandwidth* 3000 data, Algoritma *Merge Sort* memiliki waktu eksekusi yang lebih cepat dibanding kedua algoritma lainnya.

Penelitian ini menggunakan data yang masih tergolong menegah disebabkan keterbatasan *source*, memori dan *processor* yang digunakan hingga penelitian selanjutnya dapat memanfaatkan data yang berskala besar untuk mendapatkan hasil yang optimal. Selain itu pengembangan selanjutnya dapat melakukan penggabungan algoritma untuk dapat melihat kinerja dari penggabungan tersebut.

Daftar Pustaka

[1] D. Kumalasari, "Analisis Perbandingan Kompleksitas Algoritma Bubble Sort, Cocktail Sort dan Comb Sort dengan Bahasa Pemrograman C ++," *J. Speed*, vol. 9, no. 2, pp. 1–7, 2017.

[2] R. Rio Sitepu, M. Yusman, and F. Eka Febriansyah, "Implementasi Algoritma Bubble Sort Dan Selection Sort Menggunakan Arraylist Multidimensi Pada Pengurutan Data Multi Prioritas," *J. Komputasi*, vol. 5, no. 1, pp. 81–87, 2017.

[3] S. K. Neelam Yadav, "Sorting Algorithms," *Int. Res. J. Eng. Technol.*, vol. 3, pp. 425–446, 2016.

[4] N. Fadhillah, Huzain Azis, and D. Lantara, "Validasi Pencarian Kata Kunci Menggunakan Algoritma Levenshtein Distance Berdasarkan Metode Approximate String Matching," *Pros. Semin. Nas. Ilmu Komput. dan Teknol. Inf.*, vol. 3, no. 2, pp. 3–7, 2018.

[5] P. Prajapati, N. Bhatt, and N. Bhatt, "Performance Comparison of Different Sorting Algorithms," vol. VI, no. Vi, pp. 39–41, 2017.

[6] W. Ali, T. Islam, H. U. R. Rehman, I. Ahmad, M. Khan, and A. Mahmood, "Comparison of different sorting algorithms," *Int. J. Adv. Res. Comput. Sci. Electron. Eng.*, vol. 5, no. 7, pp. 63–71, 2016.

[7] S. Subandijo, "Efisiensi Algoritma dan Notasi O-Besar," *ComTech Comput. Math. Eng. Appl.*, vol. 2, no. 2, p. 849, 2017.

[8] A. Nugroho, "Pengenalan algoritma pemrograman melalui simulasi robot," *UPN "Veteran" Yogyakarta*, vol. 2015, no. November, pp. 1–7, 2015.

[9] M. Jauhari, D. Hamidin, and M. Rahmatuloh, "Komparasi Stabilitas Eksekusi Kode Bahasa Pemrogrman .Net C# Versi 4.0.3019 Dengan Google Golang Versi 1.4.2 Menggunakan Algoritma Bubble Sort dan Insertion Sort," vol. 9, no. 1, pp. 13–20, 2017.

[10] A. H. Saptadi and D. W. Sari, "Analisis Algoritma Insertion Sort, Merge Sort Dan Implementasinya Dalam Bahasa Pemrograman C++," *J. INFOTEL - Inform. Telekomun. Elektron.*, vol. 4, no. 2, p. 10, 2016.

[11] M. Arawin, "Penerapan bubble sort dan insertion sort pada urutan mahasiswa," 2018.

[12] T. SinghSodhi, S. Kaur, and S. Kaur, "Enhanced Insertion Sort Algorithm," *Int. J. Comput. Appl.*, vol. 64, no. 21, pp. 35–39, 2013.

[13] F. E. Saputro and F. N. Khasanah, "Teknik Selection Sort dan Bubble Sort Menggunakan Borland C ++," *J. Mhs. Bina Insa.*, vol. 2, no. 2, pp. 136–145, 2018.

[14] S. Jadoon, S. Solehria, and M. Qayum, "Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study," *Int. J. Electr. …*, no. April, 2011.

[15] G. S. A.Kumar, A.Dutt, "Merge Sort Algorithm [M1]," *Commun. ACM*, vol. 15, no. 5, pp. 357–358, 2014.

[16] S. Paira, S. Chandra, and S. K. S. Alam, "Enhanced Merge Sort- A New Approach to the Merging Process," *Procedia Comput. Sci.*, vol. 93, no. September, pp. 982–987, 2016.