Jurnal Informatika Universitas Pamulang **ISSN: 2541-1004**
Penerbit: Fakultas Ilmu Komputer Universitas Pamulang **e-ISSN: 2622-4615**
Vol. 10, No. 1, Maret 2025 (7-14) **https://doi.org/10.32493/informatika.v10i1.44216**

Analisa Perbandingan Algoritma *Bubble Sort, Insert Sort,* dan *Selection Sort*

Agus Heri Yunial

Teknik Informatika, Universitas Pamulang, Jl. Raya Puspitek No.46, Buaran, Serpong, Tangerang Selatan, Banten, Indonesia, 15310
e-mail: dosen02525@unpam.ac.id

Submitted Date: April 15^{th}, 2024 Reviewed Date: June 11^{th}, 2024
Revised Date: September 9^{th}, 2024 Accepted Date: March 26^{th}, 2025

Abstract

An algorithm in a program represents the flow or sequence of that program. An optimal program must have an efficient algorithm, where efficiency is measured based on minimal time complexity in data processing. A sorting algorithm is a type of algorithm that performs ordering operations within a program. Some common sorting algorithms include Bubble Sort, Insertion Sort, and Selection Sort. This study compares the processing time of these three algorithms in sorting datasets ranging from 100 to 10,000 randomly generated numbers in Excel using the RAND() function, both in ascending and descending order. The key difference between this study and previous ones is that the datasets are divided into 15 different combinations, each with varying sizes and order patterns. This allows for an in-depth comparison of how each algorithm performs across different sorting scenarios. The dataset combinations include 100, 500, 1,000, 5,000, and 10,000 random numbers, as well as partially sorted datasets, such as 50 sorted + 50 random, 250 sorted + 250 random, up to 8,000 sorted + 2,000 random numbers. The sorting process is implemented using the C++ programming language. The test results indicate that Bubble Sort has the longest execution time compared to Insertion Sort and Selection Sort in almost all cases, especially for larger datasets. In fully random datasets, Bubble Sort's execution time increases significantly as the number of elements grows, with the highest recorded time reaching 875,562,800 nanoseconds for 10,000 random data points (Type 5). For datasets that are partially sorted, Insertion Sort demonstrates better time efficiency than Bubble Sort, particularly in nearly sorted data. This is evident as Insertion Sort records execution times of 63,845,900 nanoseconds and 326,498,100 nanoseconds, which are faster than Bubble Sort's 139,313,800 nanoseconds and 547,248,700 nanoseconds. Meanwhile, Selection Sort generally performs faster than Bubble Sort in all scenarios and, in some cases, even outperforms Insertion Sort, particularly for large random datasets. However, in nearly sorted datasets, Insertion Sort remains the fastest among the three, demonstrating its advantage in handling partially ordered data efficiently.

Keywords: *sorting* algorithm, *bubble sort, insert sort, selection sort*

Abstrak

Algoritma pada sebuah program menggambarkan alur atau urutan dari program tersebut. Program yang optimal harus memiliki algoritma yang efisien, sehingga efisiensi algoritma dapat diukur berdasarkan kompleksitas waktu yang minimal dalam pemrosesan datanya. Algoritma *sorting* adalah algoritma yang melakukan proses pengurutan dalam program tersebut. Beberapa program *sorting* yang ada di antaranya *bubble sort, insert sort,* dan *selection sort*. Penelitian ini membandingkan waktu pemrosesan ketiga algoritma tesebut dalam mengurutkan 100 hingga 10.000 data yang dibuat secara *random* di excel dengan menggunakan fungsi RAND() baik secara *ascending* maupun *descending*. Yang menjadi pembeda penelitian ini dengan penelitian sebelumnya adalah data yang disusun dibedakan menjadi 15 data kombinasi dengan jumlah dan urutan data yang berbeda, sehingga setiap algoritma akan dilihat kecepatannya untuk pengurutan semua kombinasi tersebut. Kombinasi tersebut terdiri dari 100, 500, 1.000, 5.000, dan 10.000 data acak, serta kombinasi data terurut sebagian, seperti 50 data terurut + 50 data *random*, 250 data terurut

+ 250 data *random*, hingga 8.000 data terurut + 2.000 data *random*. Proses pengurutan dilakukan dengan menggunakan bahasa pemrograman c++. Dari hasil pengujian didapatkan bahwa algoritma *Bubble Sort* memiliki waktu eksekusi yang paling lama dibandingkan dengan *Insertion Sort* dan *Selection Sort* dalam hampir semua kasus, terutama pada dataset yang lebih besar. Pada data yang sepenuhnya acak waktu eksekusi *Bubble Sort* meningkat secara signifikan seiring bertambahnya jumlah data, dengan waktu tertinggi pada 10.000 data *random* (Tipe 5) mencapai 875.562.800 *nanosecond*s. Untuk *dataset* yang memiliki sebagian data sudah terurut, *Insertion Sort* menunjukkan keunggulan dalam efisiensi waktu dibandingkan *Bubble Sort*, terutama pada dataset yang hampir terurut. Hal ini terlihat bahwa *Insertion Sort* memiliki waktu 63.845.900 *nanosecond*s dan 326.498.100 *nanosecond*s, lebih cepat dibandingkan *Bubble Sort* yang membutuhkan 139.313.800 *nanosecond*s dan 547.248.700 *nanosecond*s. Sementara itu, *Selection Sort* secara umum memiliki waktu eksekusi yang lebih cepat dibandingkan *Bubble Sort* dalam semua skenario, dan pada beberapa kasus bahkan lebih baik daripada *Insertion Sort*, seperti pada dataset *random* ukuran besar. Namun, pada data set yang hampir terurut, *Insertion Sort* tetap lebih unggul dalam kecepatan dibandingkan *Selection Sort*.

Keywords: algoritma *sorting*, *bubble sort*, *insert sort*, *selection sort*

1. Pendahuluan

Algortima bisa diartikan sebagai sebuah alur atau tatacara ataupun pedoman. Contohnya, dalam kehidupan sehari-hari algoritma memasak mie instan. Langkah pertama yang dilakukan adalah memasak air sampai mendidih, kemudian memasukan mie ke dalam air tersebut, selanjutnya menuangkan bumbu mie instan dan terkahir dihidangkan ke dalam mangkuk. Tentu saja setiap orang ada yang memiliki algoritma memasak mie yang sama dengan di atas, atau banyak juga yang memiliki algoritma yang berbeda. Misalnya, seseorang ingin membuang kuah rebusan terlebih dahulu, baru menggantinya dengan air panas. Hal itu bisa saja terjadi sesuai algoritma yang dipilih dengan kebutuhan atau tingkat keefisiean waktu yang digunakan.

Perubahan digital yang masif menjadi faktor penting akan peralihan penyimpanan data secara digital baik itu *offline* maupun online yang disimpan dalam *cloud computing*. Berbagai bidang banyak memanfaatkan penyimpanan *cloud computing* tidak terlepas dari dunia pendidikan (Kurniawan, Wiranata, Kusnan, Ma'muriyah, & Ting, 2023). Semakin banyaknya penyimpanan data secara digital, maka semakin sulit melakukan proses pencarian data yang diiginkan (Wijaya, Fauziah, & Harjanti, 2024).

Algoritma pada sebuah program menggambarkan alur atau urutan dari jalannya program tersebut. Program yang baik harus memiliki algoritma yang baik pula. Sebelum menggunakan algoritma, ada 3 hal penting yang harus dilihat yaitu algoritma harus memberikan luaran yang sesuai dari inputan, algoritma harus mendekati kebenaran, dan algoritma harus efisien (Retta, Isroqmi, & Nopriyanti, 2020). Algoritma yang efisien salah satunya adalah penggunaan waktu terbaik dalam pemrosesan program tersebut.

Algoritma pemrograman adalah serangkaian langkah logis yang disusun secara sistematis untuk menyelesaikan permasalahan dalam pemrograman komputer. Dalam proses pengembangan program, algoritma sering kali menjadi komponen awal yang harus dirancang sebelum proses penulisan kode dilakukan, karena algoritma berperan sebagai fondasi utama dari logika program (Anggreani S. & Yahfizham, 2024).

Salah satu penggunaan algortima pada pemrograman adalah algoritma pengurutan atau *sorting*. Pengurutan data merupakan salah satu proses dasar yang banyak digunakan dalam berbagai aplikasi komputer, seperti proses pencarian, penyimpanan, hingga pengolahan data. Dalam bidang pemrograman, algoritma pengurutan memiliki peranan penting untuk menjamin efisiensi akses maupun pemrosesan data yang tersedia (Yanti & Eriana, 2024).

Bubble Sort adalah salah satu metode pengurutan data. Inti dari algoritma ini terletak pada proses perbandingan antara elemen-elemen dalam *array*, di mana jika urutannya tidak sesuai, maka kedua elemen tersebut akan ditukar. Proses ini terus diulang hingga seluruh elemen berada dalam urutan yang benar dan tidak diperlukan lagi pertukaran. Karena menggunakan proses perbandingan antar elemen, algoritma ini tergolong dalam kategori *comparison sort* (Panggabean, Htb, Perina, Toro, & Syahputra, 2023) .

Insert Sort merupakan salah satu algoritma pengurutan sederhana yang bekerja dengan memulai proses dari elemen kedua dalam *array*. Elemen ini dibandingkan dengan elemen sebelumnya, dan jika nilainya lebih kecil, maka keduanya akan ditukar. Langkah serupa diterapkan secara berurutan ke elemen-elemen berikutnya, hingga seluruh data dalam *array* tersusun dalam urutan yang benar (Hakim, Ryan, Ismallah, Firdaus, & Suharsono, 2024).

Selection Sort merupakan salah satu teknik pengurutan data yang bekerja dengan cara membandingkan elemen saat ini dengan seluruh elemen setelahnya. Jika ditemukan elemen dengan nilai yang lebih kecil, posisinya akan dicatat dan kemudian dilakukan pertukaran dengan elemen awal. Proses ini diulang hingga seluruh elemen berada pada posisi yang benar dalam urutan yang diinginkan (Sandria, Nurhayoto, Ramadhani, Harefa, & Syahputra, 2023).

Penelitian yang dilakukan oleh Dhamma mengedepankan bahwa algoritma *Insertion Sort* dan *Selection Sort* memiliki kompleksitas waktu dalam kasus terburuk (Worst Case) adalah $O(n^2)$ artinya jumlah operasi tumbuh secara kuadratik seiring bertambahnya ukuran data. Jika n = 100, maka jumlah operasi kira-kira 10,000 (100^2). Namun, untuk algoritma penyisipan, kinerjanya bisa lebih optimal dengan kompleksitas O(n) jika data yang diurutkan sudah hampir teratur (Dhamma, 2023).

Insertion Sort, *Selection Sort*, dan *Bubble Sort* kurang efisien karena memiliki kompleksitas waktu rata-rata dan terburuk $O(n^2)$. Artinya, saat jumlah data besar, proses *sorting* akan menjadi sangat lambat, sehingga algoritma ini lebih cocok untuk dataset kecil atau yang hampir terurut. *Shell Sort* menawarkan peningkatan performa dengan kompleksitas rata-rata O(n log n), meskipun dalam kasus terburuk masih bisa mencapai $O(n^2)$. Ini menjadikannya lebih cepat dibandingkan dengan metode *sorting* sederhana dalam banyak kasus. Di sisi lain, *Quick Sort* dikenal sebagai salah satu algoritma *sorting* tercepat dengan kompleksitas rata-rata O(n log n). Namun, dalam skenario terburuk, jika partisi tidak optimal, kompleksitasnya bisa memburuk menjadi $O(n^2)$. Sementara itu, *Heap Sort* menawarkan stabilitas dengan kompleksitas O (n log n) baik dalam rata-rata maupun kasus terburuk, menjadikannya pilihan yang lebih andal untuk dataset besar (Li, 2024).

Penelitian berikutnya dilakukan oleh Angraeni, dkk. yang melakukan perbandingan algortima *Insertion Sort*, *Selection sort*, dan *Merge Sort*. Penelitian ini membandingkan waktu pemrosesan dan memori yang digunakan dengan memastikan jumlah masukkan data setiap algoritma yang digunakan. Data yang digunakan adalah data berbentuk tipe data *double* dari penggunaan *bandwidth* jaringan Ukhuwah NET yang terhubung di Fakultas Ilmu Komputer. Hasil penelitian ini didapatkan bahwa algoritma *Merge Sort* memiliki waktu eksekusi yang lebih cepat dengan nilai rata-rata waktu pengurutan data sebesar 108.593777 ms pada 3000 data. Sedangkan untuk waktu eksekusi paling lama adalah algoritma *Selection sort* dengan besar waktu pengurutan data 144.498144 ms dalam jumlah data yang sama, adapun dari sisi penggunaan memori dengan jumlah data 3000 algoritma *Insertion Sort* dan *Selection sort*, secara berturut-turut memiliki penggunaan memori sebesar 20.837 MB dan 20.325 MB, sedangkan Algoritma *Merge Sort* memiliki penggunaan memori yang paling tinggi dibanding kedua algoritma lainnya yaitu sebesar 21.444 MB (Anggreani, Wibawa, Purnawansyah, & Herman, 2020).

Penelitian selanjutnya dilakukan Yayan, dkk. yang membandingkan algoritma *Insertion sort*, *quick sort*, *bubble sort*, *selection sort*, dan *merge sort* dengan data sebanyak 200 sampai 500 data yang berupa data *random* bertipe interger dengan bahasa pemrograman python. Dalam proses pengurutan, dibandingkan waktu yang diperlukan dan ukuran memori yang dipakai dalam setiap algortima. Algortima yang efektif adalah algoritma yang memiliki tingkat efisiensi waktu proses yang singkat dan penggunaan memory yang sedikit. Hasil untuk efisiensi waktu algoritma *Quick Sort* lebih unggul yaitu diperlukan waktu proses 0,001 s, 0,001 s, 0,001 s, untuk inputan 200,300 dan 400 data, 0,003 s untuk inputan 500 data, dan untuk penggunaan memory, algoritma *Bubble sort* lebih unggul karena hanya memerlukan ukuran memory yang sedikit dibandingkan dengan yang lainnya dengan inputan 200, 300, 400 dan 500 diperlukan memory sebesar 3.83kB, 4.83 kB, 5,83 kB, dan 6,83 kB (Heryanto, Fauziah, & Harjanti, 2023).

Selanjutnya, penelitian dilakukan oleh Joko, dkk. yang membandingkan algortima *bubble*, *selection*, *Insertion*, *shell*, *merge* dan *quick sort* dengan menggunakan 1000 sampai 200.000 data *random*. Bahasa pemrograman yang digunakan adalah java dan python. Hasil penelitian mereka

didapat bahwa algoritma *bubble sort*, *selection sort* dan *Insertion sort* untuk jumlah data 20.000 ke atas lebih lambat dengan menggunakan bahasa Python dibanding dengan bahasa pemrograman java, sedangkan algoritma *shell sort*, *merge sort* dan *quick sort* untuk data 20.000 ke atas menghasilkan waktu yang tidak jauh berbeda untuk kedua bahasa pemrograman tadi. Hal itu terlihat pada pada hasil pengujian untuk jumlah data sebanyak 20.000, terlihat bahwa waktu eksekusi algoritma pada bahasa pemrograman Java lebih cepat dibandingkan Python. Pada Java, *Bubble Sort* membutuhkan waktu 0.473316 detik, *Selection Sort* 0.086414 detik, *Insertion Sort* 0.182907 detik, *Shell Sort* 0.008212 detik, *Merge Sort* 0.006527 detik, dan *Quick Sort* 0.003068 detik. Sementara itu pada Python, *Bubble Sort* memerlukan waktu 24.83016 detik, *Selection Sort* 7.074612 detik, *Insertion Sort* 11.52913 detik, *Shell Sort* 0.075249 detik, *Merge Sort* 0.060199 detik, dan *Quick Sort* 0.038126 detik. Dari data ini, dapat disimpulkan bahwa algoritma *Quick Sort* memiliki performa terbaik pada kedua bahasa pemrograman, dengan waktu eksekusi tercepat, sedangkan *Bubble Sort* menunjukkan performa paling lambat. Selain itu, perbedaan signifikan dalam waktu eksekusi menunjukkan bahwa Java lebih efisien dalam menangani proses *sorting* dibandingkan Python, terutama untuk jumlah data yang lebih besar. (Iskandar, Suhendar, & Pamungkas, 2024).

Pada penelitian-penelitan di atas, data yang diuji merupakan data yang seluruhnya tidak terurut atau *random*. Untuk itu pada penelitian ini data yang ingin dibandingkan adalah berupa data integer yang memilik 15 kombinasi, baik data tidak terurut sampai data yang seluruhnya hampir berurutan.

Penelitian ini merupakan perbandingan algoritma *sorting* dengan kombinasi urutan angka berbeda yang belum digunakan oleh penelitian-penelitan sebelumnya. Terdapat beberapa penelitian sebelumnya dengan menggunakan beberapa metode algoritma *sorting* yang mengurutkan beberapa angka maupun huruf.

Algoritma *sorting* dalam pemrograman ada banyak di antaranya *bubble sort*, *insert sort*, *selection sort*, *merger sort*, *quick sort*, *shell sort* dan lain sebagainya. Penelitian ini membatasi pembahasan algortima *sorting* yang dibandingkan yaitu hanya membandingkan algoritma *bubble sort*, *insert sort* dan *selection sort*. Ketiga algoritma tersebut memiliki kecepatan proses pengurutan yang berbeda-beda, namum memiliki kompleksitas waktu yang sama yaitu $O(n^2)$.

Tujuan dari penelitian ini adalah untuk mengukur algoritma mana yang paling efisien untuk mengurutkan berbagai macam data, baik itu data dalam jumlah kecil sampai data dalam jumlah besar, dan data yang hampir terurut maupun data yang sepenuhnya tidak terurut.

2. Metode Penelitian

Data yang digunakan pada penelitian ini berupa 15 kombinasi urutan data yang dibuat secara *random* di excel yang nantinya data tersebut yang akan di proses di program *sorting*. Untuk tipe kombinasi yang digunakan adalah sebagai berikut:

a. Tipe 1. 100 Data *Random*
b. Tipe 2. 500 Data *Random*
c. Tipe 3. 1000 Data *Random*
d. Tipe 4. 5000 Data *Random*
e. Tipe 5. 10.000 Data *Random*
f. Tipe 6. 50 Data urut dan 50 *Random*
g. Tipe 7. 250 Data urut dan 250 *Random*
h. Tipe 8. 500 Data urut dan 500 *Random*
i. Tipe 9. 2500 Data urut dan 2500 *Random*
j. Tipe 10. 5000 Data urut dan 5000 *Random*
k. Tipe 11. 75 Data urut dan 25 *Random*
l. Tipe 12. 400 Data urut dan 100 *Random*
m. Tipe 13. 800 Data urut dan 200 *Random*
n. Tipe 14. 4000 Data urut dan 1000 *Random*
o. Tipe 15. 8000 Data urut dan 2000 *Random*.

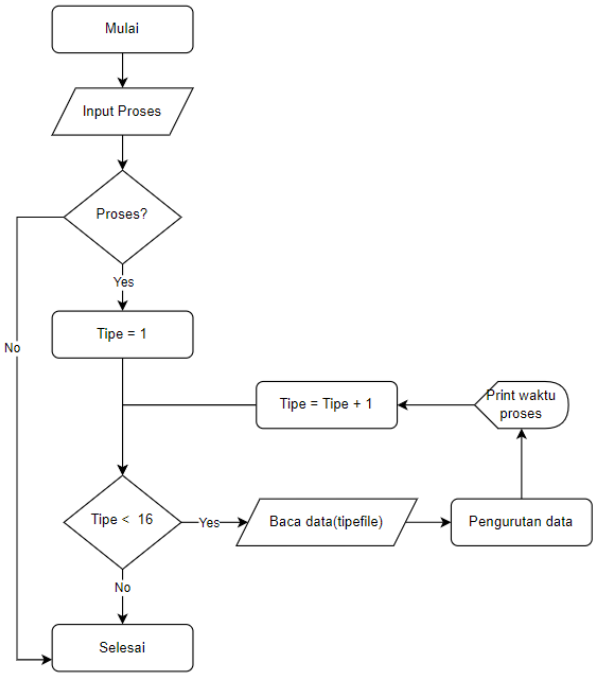
Data tersebut dibuat dalam excel yang disimpan di dalam komputer yang nantinya akan digunakan oleh program. Proses pengurutan data di excel penggunakan fungsi RAND() dan disimpan menjadi 15 file sesuai dengan 15 tipe kombinasi pengurutan yang digunakan.

Bahasa pemrograman yang digunakan adalah bahasa C++ dengan menggunakan aplikasi devC++ versi 5.11. Perangkat yang digunakan adalah laptop dengan sistem operasi windows 11 dengan 16 GB ram dan 512GB SSD yang menggunakan prosesor intel i5. Penelitian ini berfokus untuk membandingkan waktu proses pengurutan oleh algortima *bubble sort*, *insert sort* dan *selection sort* baik secara ascending maupun descending.

Perhitungan waktu proses pengurutan yang dilakukan pada penelitian ini yaitu dimulai dari program melakukan proses pengurutan data yang sudah diterima dari excel tadi sampai program selesai melakukan proses pengurutan.

Berikut adalah gambar *flowchart* dari program yang dibuat dalam melakukan perbandingan waktu antar algoritma *sorting bubble*

sort, *insert sort* dan *selection sort* baik ascending maupun descending.



Gambar 1. Flowchart program *sorting*

Pada gambar 1 di atas, menggambarkan proses pengolahan data dengan membaca file, melakukan pengurutan data, dan mencetak waktu prosesnya. Proses dimulai dengan menerima input terkait konfirmasi proses dilanjutkan atau tidak. Jika tidak, program langsung selesai. Jika ya, maka variabel Tipe diinisialisasi dengan nilai 1, dan program akan melakukan loop selama Tipe masih kurang dari 16. Dalam setiap iterasi, program membaca data dari file sesuai dengan tipe data, misal tipe 1, maka file yang dibaca adalah "data1" dan seterusnya, kemudian melakukan proses pengurutan, dan mencetak waktu eksekusi. Setelah itu, nilai Tipe ditambah 1 dan proses diulang hingga semua tipe data selesai diproses. Jika Tipe mencapai 16, maka program berhenti dan proses selesai. Dengan struktur ini, program dapat mengurutkan dan mengevaluasi performa pengurutan pada berbagai kombinasi data secara otomatis.

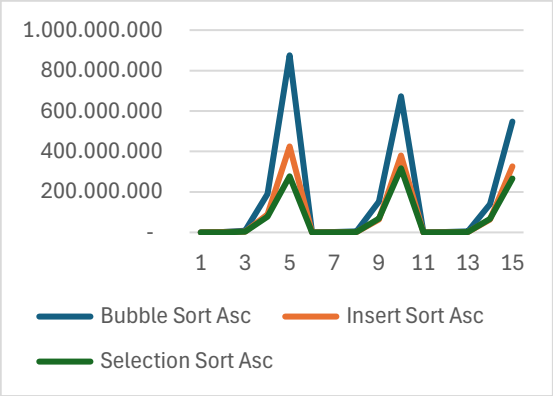
3. Hasil

Hasil output program untuk waktu pengurutan ascending dapat dilihat pada tabel 1 dan gambar 2. Untuk waktu yang dihasilkan adalah dalam *nanoseconds*.

Tabel 1. Perbandingan pengurutan secara ascending (dalam *nanoseconds*)

Tipe	Bubble Sort Asc	Insert Sort Asc	Selection Sort Asc
1	-	-	-
2	2,285,300	1,572,100	1,001,700
3	8,493,200	3,992,400	4,001,600
4	190,512,000	89,654,200	77,099,400
5	875,562,800	424,941,000	276,665,200
6	-	-	-
7	2,000,000	998,300	999,300
8	5,041,300	1,995,000	2,001,300
9	151,901,200	63,061,800	67,605,700
10	672,818,900	379,847,700	316,908,200
11	-	-	-
12	1,998,500	959,900	1,003,700
13	5,436,700	2,053,300	2,489,000
14	139,313,800	63,845,900	66,796,700
15	547,248,700	326,498,100	265,722,500

Dari tabel 1 di atas, bisa dilihat grafiknya pada gambar 2 di bawah ini:



Gambar 2. Perbandingan pengurutan secara ascending

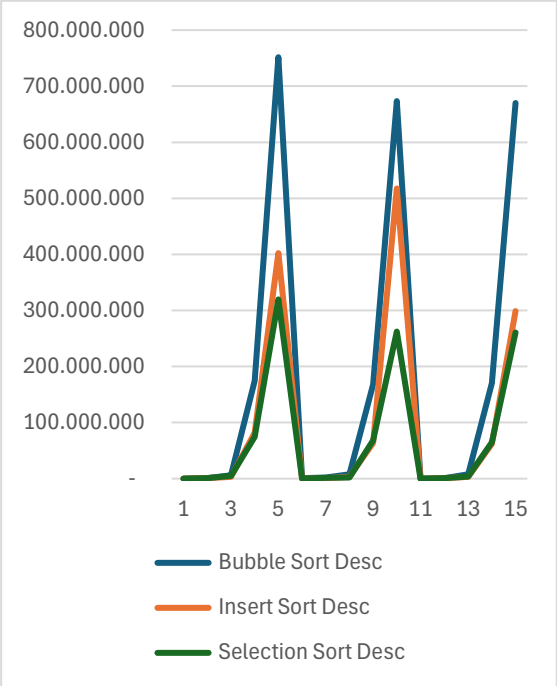
Berdasarkan data dalam tabel 1 dan gambar 2, dapat disimpulkan bahwa algoritma *Bubble Sort* memiliki waktu eksekusi yang paling lama dibandingkan dengan *Insertion Sort* dan *Selection Sort* dalam hampir semua kasus, terutama pada dataset yang lebih besar. Pada data yang sepenuhnya acak (Tipe 2–5), waktu eksekusi *Bubble Sort* meningkat secara signifikan seiring bertambahnya jumlah data, dengan waktu tertinggi pada 10.000 data *random* (Tipe 5) mencapai 875.562.800 *nanosecond*. Untuk dataset yang memiliki sebagian data sudah terurut (Tipe 6–15), *Insertion Sort* menunjukkan keunggulan dalam efisiensi waktu dibandingkan *Bubble Sort*, terutama pada dataset yang hampir terurut. Hal ini terlihat dari Tipe 14 dan 15, di mana *Insertion Sort* memiliki waktu 63.845.900 *nanosecond* dan 326.498.100 *nanosecond*, lebih cepat dibandingkan *Bubble Sort* yang membutuhkan 139.313.800 *nanosecond* dan 547.248.700 *nanosecond*. Sementara itu, *Selection Sort* secara umum memiliki waktu eksekusi yang lebih cepat dibandingkan *Bubble Sort* dalam semua skenario, dan pada beberapa kasus bahkan lebih baik daripada *Insertion Sort*, seperti pada dataset *random* ukuran besar (Tipe 5 dan 10). Namun, pada dataset yang hampir terurut, *Insertion Sort* tetap lebih unggul dalam kecepatan dibandingkan *Selection Sort*. Secara keseluruhan, dapat disimpulkan bahwa *Bubble Sort* adalah algoritma yang paling tidak efisien, terutama untuk dataset besar, sedangkan *Insertion Sort* lebih unggul dalam mengurutkan data yang hampir terurut. *Selection Sort* berada di tengah-tengah, dengan performa yang cukup stabil tetapi tetap kalah efisien dibandingkan *Insertion Sort* pada dataset yang hampir terurut.

Untuk waktu pengurutan secara descending dapat dilihat pada tabel 2 dan gambar 3.

Tabel 2. Perbandingan pengurutan secara descending (dalam *nanosecond*s)

Tipe	Bubble Sort Desc	Insert Sort Desc	Selection Sort Desc
1	-	-	-
2	954,200	1,003,600	1,013,800
3	5,956,400	3,006,000	4,992,700
4	174,366,700	81,743,900	74,477,300
5	751,683,900	402,029,400	319,738,500
6	902,000	.	.
7	2,017,600	1,006,400	1,607,800
8	7,495,300	3,016,300	2,040,800
9	168,351,200	63,613,400	68,233,400
10	673,512,600	517,557,600	262,580,400
11	-	-	-
12	964,700	999,300	996,900
13	7,529,800	2,996,900	3,535,400
14	170,301,500	61,790,400	64,711,100
15	669,725,200	298,877,000	260,420,200

Dari tabel 2 di atas, bisa dilihat grafiknya pada gambar 3 di bawah ini:



Gambar 3. Perbandingan pengurutan secara descending

Berdasarkan data dalam tabel 2 dan gambar 3 untuk proses *sorting* secara descending, dapat disimpulkan bahwa *Bubble Sort* kembali menjadi algoritma dengan waktu eksekusi paling lama dibandingkan dengan *Insertion Sort* dan *Selection Sort*, terutama pada dataset yang lebih besar. Pada data yang sepenuhnya acak (Tipe 2–5), waktu eksekusi *Bubble Sort* meningkat drastis seiring bertambahnya jumlah data, dengan waktu tertinggi pada 10.000 data *random* (Tipe 5) mencapai 751.683.900 *nanosecond*, jauh lebih lambat dibandingkan *Selection Sort* yang hanya membutuhkan 319.738.500 *nanosecond*. Pada dataset yang memiliki sebagian data sudah terurut (Tipe 6–15), *Insertion Sort* kembali menunjukkan keunggulannya dalam mengurutkan data yang hampir terurut. Contohnya, pada Tipe 14 dan 15, *Insertion Sort* hanya membutuhkan 61.790.400 *nanosecond* dan 298.877.000 *nanosecond*, jauh lebih cepat dibandingkan *Bubble Sort* yang membutuhkan 170.301.500 *nanosecond* dan 669.725.200 *nanosecond*. Sementara itu, *Selection Sort* memiliki performa yang cukup stabil dan umumnya lebih cepat dibandingkan *Bubble Sort* dalam semua skenario. Pada beberapa kasus, *Selection Sort* bahkan lebih efisien daripada *Insertion Sort*, terutama pada dataset *random* ukuran besar seperti Tipe 5 dan 10, di mana waktu eksekusinya lebih rendah dibandingkan *Insertion Sort*. Secara keseluruhan, *Bubble Sort* tetap menjadi algoritma yang paling tidak efisien, terutama pada dataset yang lebih besar, sesuai denhan penelitian yang dilakukan oleh Ahmad Yusuf dan Yerix Ramadhani yang menyimpulkan *Bubble Sort* merupakan salah satu algoritma pengurutan yang paling sederhana dan mudah untuk dijadikan sebagai langkah awal dalam memahami konsep dasar *sort*ing namun memiliki efisiensi yang rendah saat menangani data dalam jumlah besar, karena memiliki kompleksitas waktu sebesar $O(n^2)$ (Yusuf & Ramadhani, 2024). *Insertion Sort* unggul dalam mengurutkan data yang hampir terurut sesuai dengan penelitian yang dilakukan oleh Mulia Dharma yang menyatakan bahwa Algoritma *Insert*ion *Sort* dan *Selection Sort* memiliki tingkat efisiensi yang relatif setara. Namun, *Insert*ion *Sort* memiliki keunggulan tersendiri ketika data yang akan diurutkan sudah dalam kondisi hampir terurut, karena dapat menyelesaikan proses pengurutan dengan lebih cepat dalam situasi tersebut (Dhamma, 2023), sedangkan *Selection Sort* menawarkan keseimbangan performa yang cukup baik dalam berbagai kondisi contohnya pada penelitian yang dilakukan Agus Safrianti dan Ezwarsyah yang yang menggunakan *Selection Sort* dalam melakukan pengelolaan stok barang sehingga membantu mengembangkan sistem manajemen stok yang lebih baik dan efisien dalam pengelolaan stok barang (Safrianti & Ezwarsyah, 2024)

4. Kesimpulan

Berdasarkan hasil penelitian ini dapat disimpulkan bahwa *Selection Sort* terbukti menjadi algoritma yang paling cepat dalam banyak kondisi, baik saat mengurutkan data secara Ascending maupun Descending. Sebagai contoh, pada 100 data acak (Tipe 1 - Ascending), *Selection Sort* hanya membutuhkan waktu 1.001.700 *nano*detik, jauh lebih efisien dibandingkan *Bubble Sort* (2.285.300 *nano*detik) dan *Insertion Sort* (1.572.100 *nano*detik). Bahkan, pada dataset yang lebih besar seperti 8000 data urut & 2000 data acak (Tipe 15 - Descending), *Selection Sort* tetap unggul dengan waktu eksekusi 260.420.200 *nano*detik, mengalahkan *Insertion Sort* (298.877.000 *nano*detik) dan jauh lebih cepat daripada *Bubble Sort* (669.725.200 *nano*detik). Di sisi lain, *Bubble Sort* selalu menjadi algoritma dengan waktu eksekusi paling lama, terutama ketika menangani dataset besar dan tidak terurut. Misalnya, pada

10.000 data acak (Tipe 5 - Descending), *Bubble Sort* membutuhkan waktu 751.683.900 *nano*detik, sementara *Insertion Sort* lebih cepat dengan 402.029.400 *nano*detik, dan *Selection Sort* tetap menjadi yang tercepat dengan 319.738.500 *nano*detik. Hal ini membuktikan bahwa *Bubble Sort* sangat tidak efisien untuk dataset besar.

Sementara itu, *Insertion Sort* memiliki performa yang cukup baik ketika data sudah sebagian terurut. Sebagai contoh, pada 75 data urut & 25 data acak (Tipe 11 - Ascending), *Insertion Sort* hanya membutuhkan 959.900 *nano*detik, lebih cepat dibandingkan *Bubble Sort* (1.998.500 *nano*detik) dan hampir setara dengan *Selection Sort* (1.003.700 *nano*detik). Namun, saat digunakan untuk dataset besar yang sepenuhnya acak seperti 10.000 data *random* (Tipe 5 - Ascending), *Insertion Sort* memang lebih cepat dari *Bubble Sort* dengan 424.941.000 *nano*detik, tetapi tetap kalah dibandingkan *Selection Sort* yang hanya membutuhkan 276.665.200 *nano*detik. Secara keseluruhan, *Selection Sort* menjadi pilihan terbaik untuk mengurutkan dataset besar karena konsisten lebih cepat dibandingkan algoritma lainnya. Sementara itu, *Insertion Sort* cocok digunakan jika data sudah sebagian terurut, karena bisa lebih efisien dalam kondisi tertentu. *Bubble Sort* sebaiknya dihindari, karena selalu menjadi algoritma paling lambat, terutama saat menangani data dalam jumlah besar dan acak.

Referensi

Anggreani, D., Wibawa, A. P., Purnawansyah, & Herman. (2020). Perbandingan Efisiensi Algoritma Sorting dalam Penggunaan Bandwidth. *ILKOM Jurnal Ilmiah, 12*(2), 96-103.

Anggreani, S., & Yahfizham. (2024). Pengantar dan Pengenalan Konsep Dasar Algoritma Pemrograman. *Jurnal Pendidikan Berkarakter*, 282-294.

Dhamma, M. (2023). Analisis Kompleksitas Diantara Algoritma Insertion Sort dan Selection Sort dan Diimplemntasikan dengan Bahasa Pemograman Java. *Jurnal Nasional Informatika Dan Teknologi Jaringan*, 6-9.

Hakim, F. R., Ryan, Ismallah, H. S., Firdaus, M. L., & Suharsono. (2024). Penerapan Algoritma Insertion Sorting Terhadap Data Transaksi Saham Per Kota Di Indonesia. *Prosiding Seminar Nasional Sains dan Teknologi Seri 02* (pp. 332-341). Tangerang Selatan: Fakultas Sains dan Teknologi, Universitas Terbuka.

Heryanto, Y., Fauziah, & Harjanti, T. W. (2023). Analisis Perbandingan Ruang dan Waktu pada Algoritma Sorting Menggunakan Bahasa Pemrograman Python. *Jurnal Penerapan Sistem Informasi (Komputer & Manajemen), 4*(2), 342-347.

Iskandar, J., Suhendar, H., & Pamungkas, B. D. (2024). Analisis Strategi Algoritma Sorting Menggunakan Metode Komparatif pada Bahasa Pemrograman Java dengan Python. *G-Tech : Jurnal Teknologi Terapan, 8*(1), 104-113.

Kurniawan, S., Wiranata, W., Kusnan, Ma'muriyah, N., & Ting, V. V. (2023). Pemanfaatan Komputasi Awan (Cloud Computing) Pada Bidang Pendidikan. *Journal of Information System and Technology*, 403-405.

Li, M. (2024). Balancing Performance Trade-offs in Modern Sorting Methodologies. *The International Conference on Communication, Internet of Things, Automation and Architecture*, (pp. 588-603). Hunan.

Panggabean, A. B., Htb, R. R., Perina, I., Toro, Y. L., & Syahputra, A. (2023). Implementasi Algoritma Bubble Sort pada Sistem Pelayanan Perpustakaan Menggunakan Laravel. *Sudo Jurnal Teknik Informatika*, 19-27.

Retta, A. M., Isroqmi, A., & Nopriyanti, T. D. (2020). Pengaruh Penerapan Algoritma Terhadap Pembelajaran Pemrograman Komputer. *Jurnal Inovasi Pendidikan Matematika, 2*(2), 126-135.

Safrianti, A., & Ezwarsyah. (2024). Optimalisasi Pengurusan Data Dengan Algoritma Selection Sort Studi Kasus Dalam Pengelolan Stok Barang. *Jurnal Sains dan Teknologi*, 29-36.

Sandria, Y. A., Nurhayoto, M. A., Ramadhani, L., Harefa, R. S., & Syahputra, A. (2023). Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP. *Hello World Jurnal Ilmu Komputer*, 190-194.

Wijaya, S., Fauziah, & Harjanti, T. W. (2024). Perbandingan Algoritma Sorting Dengan Menggunakan Bahasa Pemograman Javascript Dalam Penggunaan Waktu Komputasi Dan Penggunaan Memori. *Satuan Tulisan Riset dan Inovasi Teknologi*, 294-302.

Yanti, F., & Eriana, E. S. (2024). *Algoritma Sorting Dan Searching.* Purbalingga: Eureka Media Aksara.

Yusuf, A., & Ramadhani, Y. (2024). nalisis Algoritma Bubble Sort Ascending/Descendingdan Implementasinya Menggunakan Bahasa Pemrograman Python. *Journal of Information System and Computing)* , 53-57.