ANALISIS PERBANDINGAN ALGORITMA SORTING TERHADAP DATA NUMERIK, KARAKTER, DAN STRING DI PYTHON BERDASARKAN WAKTU, MEMORI PUNCAK, DAN PERPINDAHAN DATA

Yunianita Rahmawati, Hamzah Setiawan
Teknik Informatika, Universitas Muhammadiyah Sidoarjo
Jl. Mojopahit 666 B, Sidowayah, Sidoarjo 61215, Indonesia
yunianita@umsida.ac.id

ABSTRAK

Penelitian ini membandingkan kinerja algoritma sorting secara sistematis melalui pengujian tiga jenis data, yaitu numerik, karakter, dan string. Algoritma sorting yang dibandingkan dalam penelitian yaitu bubble sort, shell sort, bucket sort, cocktail sort, comb sort, gnome sort, heap sort, insertion sort, introsort, merge sort, pigeonhole sort, quicksort, radix sort, selection sort, smoothsort, timsort, counting sort, cycle sort, pancake sort, bitonic sort, dan flashsort. Kinerja setiap algoritma kemudian dievaluasi menggunakan tiga metrik inti, yaitu waktu eksekusi (ms), *peak additional memory*, serta jumlah operasi perpindahan data (*moves/writes*). Untuk memastikan pengukuran waktu eksekusi lebih stabil, setiap skenario didahului *warm-up* sebanyak tiga kali yang tidak dicatat, kemudian dilakukan pengukuran waktu sebanyak 10 kali; untuk setiap pengukuran. Hasil pengujian menunjukkan bahwa algoritma terbaik adalah algoritma counting sort, karena memiliki *time m*ean paling rendah di berbagai dataset, terutama dalam data karakter (A ≈ 4.8) dan buku (A ≈ 2.3), dan tetap rendah terhadap numerik (A ≈ 6.8) serta kota (A ≈ 7.9). Selain itu juga memiliki waktu dengan STD kecil terutama dalam data numerik (B ≈ 3.3) dan kota (B ≈ 3.7), dan tetap rendah terhadap karakter (B ≈ 4.3) serta kota (B ≈ 4.4).

Kata kunci : *algoritma sorting, bubble sort, perbandingan, python, quicksort, radix sort*

1. PENDAHULUAN

Sorting merupakan proses menyusun sekumpulan elemen ke dalam urutan tertentu sesuai kriteria yang ditetapkan. Karena tersedia beragam algoritma pengurutan dengan karakteristik kinerja yang berbeda, pengembang kerap menghadapi ketidakpastian dalam menentukan algoritma yang paling sesuai ketika data yang diolah memiliki variasi karakteristik. Pernyataan ini diperkuat oleh [1] yang menegaskan bahwa banyak pengembang mengalami keraguan saat memilih teknik pengurutan yang paling efisien untuk kebutuhan tertentu, terutama ketika mengalami waktu yang terbatas. Dalam praktik pemrograman, pengurutan sering diperlakukan sebagai prosedur "standar" sehingga pemilihan algoritma cenderung dilakukan secara intuitif, padahal kinerja setiap algoritma dapat berubah bergantung ukuran data, tingkat keterurutan awal, serta sifat elemen yang dibandingkan. Akibatnya, pengguna hanya berfokus terhadap keluaran akhir yang sudah terurut tanpa menelaah biaya komputasi yang muncul selama proses berlangsung. Padahal, indikator seperti jumlah operasi perbandingan dan frekuensi pertukaran elemen merupakan komponen penting yang menentukan efisiensi sekaligus menjelaskan mengapa suatu algoritma dapat lebih unggul atau justru lebih lambat dari kondisi data tertentu.

Uji coba memanfaatkan bahasa pemrograman Python karena Python menyediakan mekanisme pengurutan bawaan yang terdokumentasi dengan jelas melalui algoritma TimSort serta bersifat adaptif terhadap data yang telah memiliki pola keterurutan. Karakteristik tersebut menjadikan Python relevan untuk pengujian empiris menggunakan data numerik, string, karakter, dan simbol. Pernyataan ini selaras dengan [2] ang membandingkan TimSort milik Python dengan algoritma pengurutan klasik terhadap data numerik.

Dengan demikian, penelitian ini merumuskan permasalahan sebagai kebutuhan untuk mengukur kinerja algoritma sorting secara sistematis melalui pengujian tiga jenis data, yaitu numerik, karakter, dan string. Analisis ketiga data tersebut diperlukan karena performa algoritma sorting tidak hanya ditentukan oleh ukuran data (n), tetapi juga oleh tipe data dan karakteristik representasinya. Data numerik (integer) umumnya memiliki perbandingan yang relatif sederhana dan mendukung algoritma berbasis digit atau rentang seperti radix, bucket, dan pigeonhole, sehingga menjadi baseline penting untuk melihat efisiensi algoritma non-comparison dibanding algoritma comparison-based. Data karakter (unicode) membawa tantangan tambahan berupa keragaman kode karakter dan distribusi nilai yang luas, sehingga dapat memengaruhi pola perbandingan serta kebutuhan penataan elemen, terutama algoritma yang melakukan banyak operasi perpindahan. Sementara itu, data string (nama kota dan judul buku) merepresentasikan kasus dunia nyata yang lebih kompleks karena melibatkan panjang string yang bervariasi serta tanda baca, yang secara komputasi lebih mahal dibanding angka. Dengan demikian, ketiga tipe data tersebut dipilih untuk memastikan hasil perbandingan mencerminkan kondisi yang beragam dan realistis.

Selain itu, pengukuran kinerja algoritma sorting secara sistematis juga melalui pengujian tiga metrik berupa waktu, memori puncak, dan perpindahan data.

Penggunaan ketiga metrik tersebut karena dapat menangkap aspek kinerja yang saling melengkapi dan relevan untuk spektrum algoritma yang diuji [3]. Waktu eksekusi menunjukkan efisiensi aktual dan dampak implementasi dalam Python, tetapi waktu saja sering tidak cukup untuk menjelaskan *trade-off* antaralgoritma. *Peak additional memory* penting karena beberapa algoritma cepat justru mengorbankan memori ekstra, misalnya merge sort dan timsort yang lazim menggunakan struktur bantu, atau bucket/radix/pigeonhole yang dapat memerlukan alokasi tambahan bergantung terhadap mekanisme pengelompokan dan rentang nilai. Sementara itu, data movement (*moves/writes/assignments*) dipilih sebagai metrik operasi yang lebih universal, karena tidak semua algoritma melakukan pertukaran elemen secara eksplisit, algoritma seperti merge sort, timsort, bucket sort, radix sort, dan pigeonhole sort cenderung melakukan penulisan ulang elemen ke buffer/array bantu sehingga swap bisa terlihat kecil atau nol, padahal biaya penataan data besar. Sebaliknya, algoritma in-place seperti bubble, insertion, selection, gnome, cocktail, dan comb umumnya menghasilkan perpindahan elemen yang tinggi, yang berkontribusi dalam waktu eksekusi. Dengan tiga metrik ini, perbedaan karakter algoritma—mulai dari kelompok $O(n^2)$ (bubble, insertion, selection, gnome, cocktail, comb); kelompok O(n log n) (heap, quick, merge, intro, smoothsort, timsort, shell); hingga kelompok non-comparison yang bergantung dalam distribusi nilai (bucket, radix, pigeonhole)—dapat dijelaskan secara lebih adil, tidak semata-mata berdasarkan kecepatan, tetapi juga biaya memori dan intensitas penataan data.

Algoritma sorting mampu merepresentasikan spektrum pendekatan sorting yang luas, sehingga perbandingan kinerja dapat dilakukan secara lebih komprehensif dan tidak bias terhadap satu kelompok algoritma tertentu. Pertama, kumpulan algoritma yang diuji mencakup comparison-based sorting dan non-comparison sorting; hal ini memungkinkan penelitian mengidentifikasi perbedaan karakter dasar antara dua paradigma, yakni algoritma yang mengandalkan operasi perbandingan (misalnya bubble, insertion, selection, quick, heap, merge, timsort, introsort, smoothsort, dan shell) dan algoritma yang memanfaatkan mekanisme pengelompokan (misalnya radix, bucket, dan pigeonhole), yang dalam kondisi tertentu berpotensi lebih efisien untuk data numerik diskrit. Kedua, algoritma yang dipilih juga merepresentasikan variasi kompleksitas waktu dan strategi pengolahan data, mulai dari algoritma berkompleksitas kuadratik yang umum digunakan sebagai baseline pembelajaran dan pembanding (misalnya bubble, insertion, selection, gnome, cocktail, dan comb), hingga algoritma yang lebih efisien dan lazim diterapkan dalam skala data besar (misalnya quick, merge, heap, timsort, introsort, dan smoothsort), serta algoritma yang kinerjanya sangat dipengaruhi distribusi nilai (misalnya radix, bucket, dan pigeonhole). Ketiga, keberagaman mekanisme internal—misalnya pendekatan in-place dibanding penggunaan struktur bantu, serta operasi berbasis pertukaran dibanding penulisan ulang elemen—menjadikan algoritma-algoritma tersebut relevan untuk dianalisis menggunakan metrik waktu eksekusi, peak additional memory, dan operasi perpindahan data, karena ketiga metrik tersebut mampu mengungkap trade-off kinerja dan kebutuhan sumber daya yang berbeda dalam tiap algoritma untuk berbagai tipe data yang diuji.

2. TINJAUAN PUSTAKA

Studi mengenai perbandingan algoritma sorting telah banyak dilakukan. Ringkasan pemetaannya disajikan dalam Tabel 1, dimana studi-studi sebelumnya dipetakan berdasarkan bahasa pemrograman yang digunakan. Pemetaan ini bertujuan untuk mengidentifikasi kesenjangan analisis penelitian ini. Berdasarkan pemetaan penelitian terdahulu mengenai perbandingan algoritma sorting, dapat diidentifikasi bahwa jenis data yang digunakan umumnya terbatas terhadap data numerik berupa bilangan bulat dengan urutan acak dalam rentang tertentu. Bahasa pemrograman yang paling sering digunakan adalah Python, dengan metrik evaluasi yang dominan berupa waktu eksekusi untuk merepresentasikan kompleksitas algoritma, serta pengujian yang umumnya dilakukan hanya dalam satu kali iterasi. Berbeda dengan penelitian-penelitian tersebut, penelitian ini melakukan pengujian algoritma pengurutan dalam beragam tipe data, meliputi data numerik, karakter, dan string. Evaluasi kinerja algoritma tidak hanya menggunakan waktu eksekusi, tetapi juga mencakup memori puncak dan perpindahan data. Selain itu, algoritma pengurutan seperti tim sort, pigeonhole sort, smoothsort, cycle sort, pancake sort, bitonic sort, dan flashsort, belum pernah dibandingkan secara komprehensif dengan algoritma pengurutan lainnya dalam penelitian terdahulu. Oleh karena itu, kondisi tersebut menunjukkan adanya *research gap* menjadi fokus utama penelitian ini.

Algoritma sorting dapat digunakan secara luas dalam berbagai penerapan dalam dunia nyata.

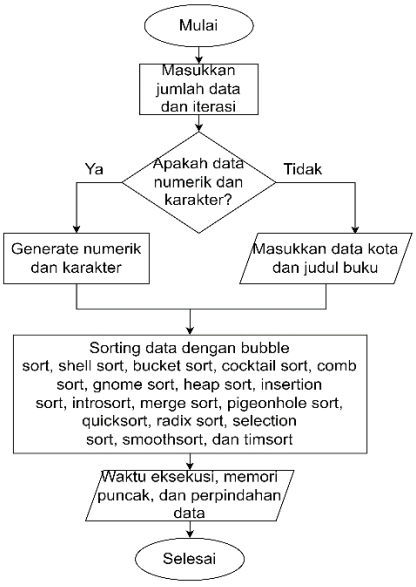
Tabel 1. Studi Mengenai Perbandingan Algoritma Sorting Berdasarkan Bahasa Pemrograman

Bahasa	Data		Algoritma Sorting	Metrik (Iterasi)
	Jenis	Batas Elemen Data		
Python	Numerik	100, 1000, 10000	*Array sorting*, bubble, shell, merge, quick, dan heap [33].	Memori dan waktu (9)
	Numerik	1000 hingga 10000	Selection, shell, dan merge, secara *ascending* dan *descending* [34].	Waktu (5)

Bahasa	Data		Algoritma Sorting	Metrik (Iterasi)
	Jenis	Batas Elemen Data		
	Numerik	100, 150, dan 300	Bubble, insertion, merge, dan quick [35].	Waktu dan kompleksitas algoritma (1)
	Numerik	10, 50, dan 100	Bubble, selection, dan insertion, secara ascending dan descending [36].	Swapping (10)
	Numerik	Iris, mahasiswa, Anggur, Uniform, Normal, Eksponensial, Bimodal, Yelp, MNIST	Quick, timsort, merge, heap, radix, dan shell [37].	Waktu eksekusi, memori, kompleksitas algoritma, dan stabilitas (1)
	Numerik	10 dan 100	Merge, heap, quick, insertion, selection, dan bubble [38].	Waktu (1)
	Numerik	10 hingga 1000000	Heap shell sort, quick, timsort, dan merge [2].	Waktu (1)
	Numerik	1000, 100000, dan 500000	Exchange sort dan insertion sort [39].	Waktu (1)
	Numerik	100, 1000, dan 10000	Array sorting, bubble, dan quick [40].	Waktu (1)
	Numerik	650	Bubble, insertion, dan selection [41]	Waktu (1)
	Numerik	100, 1000, dan 10000	Heap, quick, merge, dan shell [42]	Waktu (1)
	Numerik	ISBN GoodRead's data	Quick, Heap, Merge, Introsort, Radix [25]	Memori dan waktu (5)
Mathlab	Numerik, character	100-100.000	Merge dan quick [1].	Waktu (1)
	Numerik	1000 dan 1000000	Quick dan merge [43]	Waktu (1)
Java	Numerik	100, 1000, 10000	Bubble, insertion, selection, shell, quick, merge, heap, dan bubble [44].	Memori dan waktu (3)
	Numerik	10	Bubble, secara *ascending* dan *descending* [45]	Waktu (1)
	Numerik	50 hingga 100000	Bubble, selection, quick, dan merge [46]	Waktu (1)
	Numerik	100	Heap, insertion, dan merge [47]	Ruang dan waktu (1)
	Numerik	0 hingga 100	Bubble, shell, dan quick [48]	Waktu (1)
	Numerik	3000	Comb, cocktail, dan counting [26]	Waktu (1)
C++	Numerik	100 hingga 10000	Quick, insertion, selection, merge, dan bubble [49].	Waktu (1)
	Numerik	10000 dan 1000000	Merge, quick, heap, selection, insertion, dan bubble [50]	Memori dan waktu (1)
	Numerik	100, 1000, dan 10000	Bubble, selection, insertion, quick, merge, heap, counting, dan radix [51]	Memori dan waktu (9)
DevC++, OpenMP, C/C++	Numerik	NIM mahasiswa	Bubble dan selection [52]	Waktu dan ruang (1)
DevC++	Numerik	0 hingga 30	Bubble [53]	Waktu (1)
PHP	Numerik	1000 hingga 35000	Quick, bubble, merge, radix, dan counting [54].	Memori dan waktu (1)
C# dan Go	Numerik	1 hingga 100	Bubble Sort dan Insertion Sort [55]	Waktu (1)
JavaScript	Numerik	100, 150, dan 200	Merge, insertion, bubble, heap, quick, selection, dan shell [56]	Memori dan waktu (1)
Octave	Numerik	1000, 5000, 10000, 15000, dan 20000	Bubble, selection, insertion, merge, dan quick [57]	Waktu dan array (1)

Misalnya, untuk penentuan jarak [4], pengurutan data penjualan [5][6][7][8][9], retensi data [10], seleksi penerimaan mahasiswa [11][12][13][14], pengurutan lagu [15], pengurutan prestasi siwa [16], peringkat café (gabungan naïve bayes dan sorting) [17], efisiensi memori (gabungan ID3 dan sorting) [18], penilaian antarmuka dan pengalaman pengguna [19], penempatan pivot [20], pengurutan aset perusahaan [21], pengurutan saham [22], pengurutan KPU Blitar [23], alat bantu pembelajaran [24], dan pengurutan ISBN buku [25].

3. METODE PENELITIAN



Gambar 1. Metode Penelitian

Gambar 1 menggambarkan alur metode penelitian yang diawali dengan memasukkan parameter jumlah data dan pengaturan iterasi pengujian. Setelah itu dilakukan percabangan berdasarkan jenis data yang akan diuji. Apabila data termasuk kategori numerik atau karakter, penelitian membangkitkan (*generate*) data uji secara otomatis, yaitu data numerik berupa data acak berukuran n=12.000 bertipe integer dengan rentang nilai 1000–12.000, serta data karakter berupa Unicode *printable* berjumlah 12.000 karakter. Data string, data tidak di-*generate*, meliputi 12.000 nama kota dunia dan judul buku. Selanjutnya, seluruh data dalam setiap skenario diuji dengan serangkaian algoritma sorting yang menjadi objek penelitian, yakni bubble sort, shell sort, bucket sort, cocktail sort, comb sort, gnome sort, heap sort, insertion sort, introsort, merge sort, pigeonhole sort, quicksort, radix sort, selection sort, smoothsort, timsort, counting sort, cycle sort, pancake sort, bitonic sort, dan flashsort. Kinerja setiap algoritma kemudian dievaluasi menggunakan tiga metrik inti, yaitu waktu eksekusi (ms), *peak additional memory* sebagai indikator memori puncak selama proses pengurutan, serta jumlah operasi perpindahan data (*moves/writes/assignments*). Untuk memastikan pengukuran waktu eksekusi lebih stabil, setiap skenario didahului *warm-up* sebanyak tiga kali yang tidak dicatat, kemudian dilakukan pengukuran waktu sebanyak 10 kali; untuk setiap pengukuran. Proses pengurutan dijalankan terhadap salinan baru dari dataset yang sama agar kondisi input tetap identik. Adapun metrik memori puncak dan operasi perpindahan data dicatat satu kali per skenario karena bersifat relatif stabil dan deterministik untuk input yang sama. Proses penelitian diakhiri setelah seluruh kombinasi algoritma, tipe data, dan kondisi string selesai diuji dan seluruh metrik terekam secara lengkap.

Algoritma bubble sort, insertion sort, selection sort, shell sort, counting sort, cycle sort, cocktail sort, bucket sort, gnome sort, radix sort, heap sort, merge sort, quick sort mengacu referensi [3]. Sementara itu, algoritma intro sort dirujuk dari [25], comb sort [26], tim sort [27], smoothsort [28], pigeonhole sort [29], bitonic sort [30], pancake sort [31], serta flashsort [32]. Metrik waktu eksekusi, *peak additional memory*, dan perpindahan data mengacu [3].

4. HASIL DAN PEMBAHASAN

Hasil pengujian terhadap data numerik, karakter dan string dengan berbagai algoritma sorting disajikan di Tabel 2. Masing-masing data diuji berdasarkan waktu berupa rata-rata dan standar deviasi (STD), memori puncak, dan perpindahan data (*moves write*).

Hasil pengujian 21 algoritma sorting terhadap empat jenis dataset berupa numerik (angka bulat acak), karakter (unicode), serta string (kota dan buku). Untuk setiap jenis dataset, kinerja algoritma dinilai memakai empat metrik yang sama yaitu A = time mean (rata-rata waktu eksekusi; makin kecil makin cepat), B = time STD (stabilitas waktu; makin kecil makin konsisten), C = *peak memory* (puncak penggunaan memori), dan D = *moves/write* (jumlah operasi perpindahan atau penulisan data; makin kecil makin sedikit perpindahan). Algoritma sederhana berkompleksitas $O(n^2)$ seperti bubble, insertion, selection, cocktail, gnome, dan pancake cenderung menghasilkan time mean tinggi dan *moves/write* besar, terutama saat data berupa karakter dan buku (string atau unicode biasanya lebih "mahal" dibanding numerik). Sedangkan algoritma yang memanfaatkan teknik distribusi atau pemetaan nilai cenderung jauh lebih cepat tetapi perlu perhatian terhadap memori dan syarat rentang data. Algoritma yang paling "baik" (paling cepat dan relatif stabil lintas dataset) adalah algoritma 17 (counting sort), karena memiliki *time mean* paling rendah di berbagai dataset, terutama dalam data karakter ($A \approx 4.8$) dan buku ($A \approx 2.3$), dan tetap rendah terhadap numerik ($A \approx 6.8$) serta kota ($A \approx 7.9$). Selain itu juga memiliki waktu dengan STD kecil terutama dalam data numerik ($B \approx 3.3$) dan kota ($B \approx 3.7$), dan tetap rendah terhadap karakter ($B \approx 4.3$) serta kota ($B \approx 4.4$).

Hasil pengujian menunjukkan bahwa kinerja algoritma sorting tidak cukup dijelaskan hanya dengan satu indikator "paling cepat", karena perbandingan dilakukan terhadap empat tipe data (numerik, karakter, kota, buku) dan empat metrik kinerja, yaitu waktu eksekusi rata-rata (time mean), variasi waktu eksekusi (time STD), penggunaan *peak memory*, serta jumlah operasi perpindahan/penulisan (*moves/write*). Secara umum tampak bahwa perbedaan tipe data memengaruhi perilaku algoritma, terutama ketika operasi pembanding menjadi lebih mahal untuk data string (karakter, kota, buku), sehingga algoritma dengan kompleksitas teoretis sama dapat menghasilkan performa empiris yang berbeda terhadap skenario input yang berbeda.

Tabel 2. Hasil Pengujian Sorting

Al	Numerik				Karakter (Unicode)				String (Kota)				String (Buku)			
	A	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*	*A*	*B*	*C*	*D*
1	13927.6	2229.1	246556	71703290	12390.6	1893.5	246220	71147796	13348.7	574.1	246276	71806718	22119.8	11343.9	245660	72230516
2	6541.9	1023.7	245572	36256076	143684.8	19821.2	2074357	35693656	7033.8	12.9	245404	35915358	6315.6	1367.1	245516	36127257
3	59.6	48.9	310112	163616	81.2	35.4	310840	163616	69.9	33.2	310112	163616	66.7	42.5	310112	163616
4	26.8	3.1	288212	60000	32.9	16.4	435795	60000	51.9	27.5	779924	60000	84.4	7.3	832604	48000
5	55.0	3.6	96604	11999	71.1	32.2	96604	11999	80.5	49.7	244740	11999	85.6	55.2	96604	11999
6	4241.4	745.1	245188	36000	5358.0	623.1	245188	36000	5796.9	471.2	245244	36000	5686.8	844.9	245188	36000
7	72.8	36.4	96284	334269	58.9	27.2	96284	246731	68.0	7.4	96284	330150	92.3	81.6	96284	330985
8	9878.8	987.1	394026	36207971	10440.1	1376.5	246332	35551330	11009.5	682.3	246332	35903359	11071.6	1089.6	246332	36115258
9	17.4	58.8	1356219	12000	20.1	49.6	1083292	12000	25.2	59.6	1227731	12000	28.3	62.1	1234531	12000
10	45.9	9.4	96284	73710	52.6	23.5	96284	36150	63.8	23.8	96284	77219	74.1	53.2	96284	77676
11	26.5	21.3	98556	12000	129.9	74.7	107324	12000	34.3	13.8	98428	12000	37.9	22.5	98300	12000
12	16875.1	10388.3	246124	36101008	17471.8	9577.4	246516	35555385	19032.8	9918.9	245900	35903359	18449.5	9188.6	245452	36115258
13	625.8	98.0	249860	12000	2633.1	181.9	265380	12000	149.6	28.2	99516	12000	191.5	40.3	100156	12000
14	4.2	1.8	568200	12000	4.4	3.6	96888	12000	7.6	4.1	100604	12000	6.8	0.8	97264	12000
15	56.4	8.8	96964	12000	87.7	47.1	96916	12000	76.5	40.4	96916	12000	73.7	45.6	96916	12000
16	29.5	2.9	192684	12000	42.2	22.8	192684	12000	49.1	27.2	192684	12000	46.5	24.6	192684	12000
17	**6.8**	**3.3**	**771929**	**12000**	**4.8**	**4.3**	**195896**	**12000**	**7.9**	**3.7**	**143888**	**12000**	**2.3**	**4.4**	**112016**	**12000**
18	14808.7	3083.0	247116	11998	16289.5	898.8	247060	11880	18221.8	789.8	246364	11998	17629.5	1499.5	245996	11998
19	3185.2	280.5	288336	23973	3157.3	522.7	384568	23862	4691.7	1124.2	288336	23979	4315.6	611.4	288336	23979
20	143.5	21.2	97532	12000	158.1	64.3	245684	12000	173.4	93.7	97484	12000	164.4	76.1	97484	12000
21	12.8	18.9	864700	12000	7.5	6.7	860668	12000	17.7	11.8	788674	12000	16.9	78.2	697520	12000

Dari sisi kelas kompleksitas, terlihat pemisahan yang tegas berupa kelompok O(n^2) menghasilkan waktu eksekusi sangat besar, misalnya data numerik Bubble sort (Al=1) = 13927.6 dan Insertion sort (Al=2) = 65419.7, bahkan Cocktail sort (Al=8) = 98788.8. Pola ini konsisten dengan metrik perpindahan data yang ekstrem, misalnya *moves/write* Bubble sort dalam data numerik mencapai 71703290, sehingga pembahasan dapat menegaskan bahwa ukuran data yang diuji, algoritma O(n^2) tidak kompetitif baik dari sisi waktu maupun biaya perpindahan data.

Dalam kelompok algoritma berkompleksitas O(n log n), fokus analisis yang lebih substansial bukan sekadar menentukan algoritma tercepat, melainkan menjelaskan mengapa urutan kinerja berubah antar tipe data. Untuk data numerik, *Quick sort* (Al=11) menghasilkan *time mean* 26.5, lebih rendah daripada *Heap sort* (Al=5) 55.0 dan *Merge sort* (Al=3) 59.6. Sebaliknya, untuk data Karakter, *Quick sort* menunjukkan penurunan performa dengan *time mean* 129.9, sementara *Merge sort* lebih unggul dengan *time mean* 81.2. Perbedaan tersebut mengisyaratkan bahwa data string mempunyai performa tidak semata ditentukan oleh kompleksitas asimtotik, tetapi turut dipengaruhi biaya pembandingan, panjang token, serta pola akses memori yang meningkatkan *overhead* eksekusi. Akibatnya, algoritma yang secara teoretis berada dalam kelas kompleksitas yang sama dapat memperlihatkan perilaku empiris yang berbeda ketika operasi pembandingan menjadi lebih mahal. Pola variasi sejenis juga muncul dalam data Buku, dengan *Timsort* (Al=16) mencatat *time mean* 46.5 dan tetap kompetitif. Temuan ini menegaskan bahwa kompleksitas teoretis saja belum memadai untuk menjelaskan hasil pengujian secara komprehensif; karakteristik tipe data dan sifat operasi pembanding perlu dipertimbangkan sebagai determinan utama performa dalam implementasi nyata.

Ditinjau dari *time mean*, dominasi algoritma non-comparison paling menonjol. Pigeonhole sort menjadi yang tercepat dalam tiga skenario yaitu data Numerik = 4.2, Karakter = 4.4, dan Kota = 7.6. Data Buku, waktu terbaik justru dicapai oleh Counting sort = 2.3, lebih rendah dibanding Pigeonhole sort dalam domain yang sama (6.8). Pola ini memperlihatkan bahwa teknik non-comparison dapat mencapai efisiensi sangat tinggi ketika pemetaan kunci berjalan efektif. Sebaliknya, kelompok O(n^2) cenderung menghasilkan waktu eksekusi jauh lebih besar dalam seluruh tipe data; misalnya Bubble sort mempunyai rentang waktu tertinggi, yaitu Numerik = 13927.6, Karakter = 12390.6, Kota = 13348.7, dan Buku = 22119.8, sehingga mempertegas bahwa biaya komputasi meningkat drastis ketika jumlah operasi perbandingan dan pertukaran elemen membesar. Kelompok O(n log n) berada di tengah—lebih kompetitif daripada O(n^2), tetapi tetap kalah dari non-comparison; misalnya Quick sort untuk data Numerik (26.5) atau Timsort untuk data Numerik (29.5) masih berada di atas Pigeonhole sort (4.2).

Selain itu, *time STD (B)* memberi dasar akademik untuk membahas stabilitas kinerja (bukan stabilitas urutan sort). Pigeonhole sort tidak hanya cepat, tetapi juga relatif stabil di seluruh tipe data (STD: Numerik = 1.8, Karakter = 3.6, Kota = 4.1, Buku = 0.8). Counting sort juga berada dalam variasi yang relatif kecil dan seragam (STD: 3.3–4.4 untuk empat tipe data). Sebaliknya, algoritma berbasis perbandingan dapat menunjukkan fluktuasi yang lebih besar dalam data string; contohnya Quick sort memiliki *time STD* 21.3 untuk Numerik dan meningkat hingga 74.7 untuk Karakter, yang mengindikasikan performa yang lebih

sensitif terhadap karakteristik input ketika biaya pembandingan string menjadi dominan. Dengan demikian, tabel tidak hanya membedakan "cepat vs lambat", tetapi juga "stabil vs tidak stabil" dalam pengulangan eksperimen.

Keunggulan waktu untuk non-comparison dibayar dengan penggunaan memori puncak yang cenderung lebih besar dalam beberapa skenario; misalnya untuk data Numerik, Pigeonhole sort menggunakan *peak memory* sekitar 568200, sedangkan Quick sort sekitar 98556, dan Counting sort bahkan mencapai sekitar 771929. Namun dari sisi *moves/write*, kedua algoritma non-comparison tersebut menunjukkan nilai yang konsisten dan rendah (sekitar 12000), menandakan minimnya operasi perpindahan elemen dibanding algoritma O(n^2). Hal ini kontras dengan Bubble sort untuk data Numerik yang memiliki *moves/write* sangat besar (71703290), selaras dengan waktu eksekusi yang juga sangat tinggi.

Oleh karena itu, penelitian ini menyediakan tiga lapisan pembahasan yang lengkap yaitu (1) keunggulan non-comparison untuk *time mean*, (2) konsistensi algoritma melalui *time STD*, dan (3) konsekuensi penggunaan sumber daya melalui *peak memory* dan *moves/write*.

5. KESIMPULAN DAN SARAN

Permasalahan penelitian ini yaitu mengukur kinerja algoritma sorting secara sistematis melalui pengujian tiga jenis data, yaitu numerik, karakter, dan string. Algoritma yang paling "baik" adalah algoritma counting sort, karena memiliki *time mean* paling rendah di berbagai dataset, terutama dalam data karakter (A ≈ 4.8) dan buku (A ≈ 2.3), dan tetap rendah terhadap numerik (A ≈ 6.8) serta kota (A ≈ 7.9). Selain itu juga memiliki waktu dengan STD kecil terutama dalam data numerik (B ≈ 3.3) dan kota (B ≈ 3.7), dan tetap rendah terhadap karakter (B ≈ 4.3) serta kota (B ≈ 4.4).

Riset lanjutan dapat menguji skalabilitas melalui variasi ukuran data yang jauh lebih besar untuk melihat pola kompleksitas empiris, sekaligus menilai karakteristik distribusi data seperti kondisi hampir terurut, terbalik, banyak duplikasi, atau data string dengan panjang yang bervariasi. Selain itu, penelitian dapat menambahkan aspek stabilitas sorting (stable vs unstable) dalam skenario multi-kunci yang sering muncul dalam data teks, serta mengkaji pengaruh normalisasi unicode dan strategi pembentukan key dalam sorting string yang sangat relevan bagi data karakter atau unicode.

DAFTAR PUSTAKA

[1] S. Sundaramoorthy and G. Karunanidhi, "A Systematic Analysis on Performance and Computational Complexity of Sorting Algorithms," *Discov. Comput.*, vol. 28, no. 250, pp. 1–30, 2025.

[2] F. R. Wibowo and Muhammad Faisal, "Comparative Analysis of Sorting Algorithms : TimSort Python and Classical Sorting Methods," *JISA (Jurnal Inform. dan Sains)*, vol. 07, no. 01, pp. 11–18, 2024.

[3] M. Alaa *et al.*, "Comparative Analysis of Sorting Algorithms : A Review," *2024 11th Int. Conf. Soft Comput. & Mach. Intell.*, vol. 11, pp. 88–100, 2024.

[4] Y. P. Sari, R. Ali, and Arya Rajasa, "Perbandingan Efisiensi dengan Algoritma Sorting dalam Penentuan Jarak (Studi Kasus: Pet Shop di Bandar Lampung)," *J. Tek.*, vol. 16, no. 01, pp. 149–159, 2022.

[5] M. Rizky, S. Barokah, T. Saputra, and T. Sutabri, "Perbandingan Kinerja Algoritma Bubble Sort dan Insertion Sort dalam Pengurutan Data Penjualan UMKM," *MIFORTEKH (Jurnal Manaj. Inform. Teknol.*, vol. 5, no. 1, pp. 184–195, 2025.

[6] Z. F. Hakim, "Implementasi Metode Selection Sort Untuk Menentukan Barang Yang Harus Di Stok Ulang Dalam Sistem Informasi Penjualan," *JIEET (Journal Inf. Eng. Educ. Technol.*, vol. 01, no. 01, pp. 18–26, 2017.

[7] D. A. Saputra, M. S. Ramadhani, and M. A. Failandri, "Analisis Perbandingan Algoritma Sorting dalam Javascript untuk Meningkatkan Efisiensi Pengurutan Produk pada Aplikasi E-Commerce," *Sainstech J. Penelit. dan Pengkaj. Sains dan Teknol.*, vol. 35, no. 1, pp. 1–10, 2025.

[8] D. R. Poetra and N. Hayati, "Performa Algoritma Bubble Sort dan Quick Sort pada Framework Flutter Dan Dart SDK (Studi Kasus Aplikasi E-Commerce)," *J. Tek. Inform. dan Sist. Inf.*, vol. 9, no. 2, pp. 806–816, 2022.

[9] A. Safrianti and Ezwarsyah, "Optimalisasi Pengurusan Data Dengan Algoritma Selection Sort Studi Kasus Dalam Pengelolan Stok Barang," *J. Sains dan Teknol. 4.0 (JST 4.0)*, vol. 1, no. 2, pp. 29–36, 2024.

[10] Y. W. T. Arif, "Implementasi Metode Selection Sort pada Sistem Informasi Retensi Dokumen Rekam Medis di Klinik PKU Muhammadiyah," *J. Ilm. Elektron. DAN Komput.*, vol. 15, no. 1, pp. 108–115, 2022.

[11] Arbansyah, M. F. N. Ilham, S. H. Suryawan, and Pandu Wirayuda, "Application of Bubble Sort Optimization in New Student Admission Selection Using Brute Force Algorithm," *TEPIAN*, vol. 5, no. 2, pp. 79–86, 2024.

[12] F. Pratama, D. Juniardi, M. F. Arif, and Suharsono, "Implementasi Algoritma Selection Sort untuk Pengurutan Data Mahasiswa di Program Studi Teknik Informatika Politeknik Negeri Pontianak," *Pros. Semin. Nas. Sains dan Teknol. Seri 02*, vol. 1, no. 2, pp. 282–292, 2024.

[13] R. Nasution, A. Syahputra, A. Widiyanto, D. Subuhanto, and A. Y. Abdillah, "Persepsi Mahasiswa Informatika Terhadap Keefektifan Algoritma Bubble Sort dalam Mengurutkan

Data," *J. Tek.*, vol. 1, no. 4, pp. 1–6, 2023.
[14] Y. A. Sandria, M. R. A. Nurhayoto, L. Ramadhani, and R. S. Harefa, "Penerapan Algoritma Selection Sort untuk Melakukan Pengurutan Data dalam Bahasa Pemrograman PHP," *J. Ilmu Komput.*, vol. 1, no. 4, pp. 1–5, 2022.
[15] G. M. Putri, K. A. Di Pradja, M. B. M. Azizi, P. Nurwahid, A. S. Perdana, and Munawir, "Implementasi Stack dan Array pada Pengurutan Lagu dengan Metode Selection Sort," *J. Teknol. Dan Sist. Inf. Bisnis*, vol. 6, no. 2, pp. 286–296, 2024.
[16] E. Sunandar, "Implementasi Algoritma Bubble Sort Terhadap 2 Buah Model Varian Pengurutan Data Menggunakan Bahasa Program Java," *PETIR J. Pengkaj. dan Penerapan Tek. Inform.*, vol. 14, no. 2, pp. 159–169, 2021.
[17] N. L. Arifiyah, Gunawan, and Sawaviyya Anandianskha, "Penerapan Metode Naïve Bayes Classifier dan Selectin Sort untuk Menentukan Peringkat Cafe Di Kota Tegal," *Innov. J. Soc. Sci. Res.*, vol. 4, no. 3, pp. 17628–17641, 2024.
[18] I. D. Iskandar, I. Amirulloh, M. W. Pertiwi, M. Kusmira, A. B. Hikmah, and D. Supriadi, "Analysis Of Bubble Sort And Insertion Sort Algorithm On Memory Efficiency Using Data Mining Approach," *J. PILAR Nusa Mandiri*, vol. 16, no. 1, pp. 89–96, 2020.
[19] M. Z. Abidin, D. Nugraheny, and Y. Indrianingsih, "Comparison of Quicksort and Mergesort Method for User Interface and User Experience Assessment In," *Compiler*, vol. 9, no. 1, pp. 9–18, 2020.
[20] R. Rijaya and Muhammad Ezar Al Rivan, "Perbandingan Penempatan Pivot Pada Quick Sort Berdasarkan Ukuran Pemusatan Data," *J. Algoritm.*, vol. 4, no. 1, pp. 13–20, 2023.
[21] S. Triyono, M. F. Adithya, A. F. A. Putra, M. A. D. Husri, and Suharsono, "Visualisasi Algoritma Sorting Menggunakan Django pada Data Aset Perusahaan Berdasarkan Kapitalisasi Pasar," *Pros. Semin. Nas. Sains dan Teknol. Seri 02*, vol. 1, no. 2, pp. 293–300, 2024.
[22] F. R. Hakim, Ryan, H. S. Ismallah, M. L. Firdaus, and Suharsono, "Penerapan Algoritma Insertion Sorting Terhadap Data Transaksi Saham Per Kota di Indonesia," *Pros. Semin. Nas. Sains dan Teknol. Seri 02*, vol. 1, no. 2, pp. 332–341, 2024.
[23] N. Haming, S. Lestanti, and Saiful Nur Budiman, "Aplikasi Pengelolaan Surat Keluar Menggunakan Sequential Search Dan Selection Sort Pada Kpu Kota Blitar," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 6, no. 1, pp. 17–25, 2022.
[24] P. D. Setyorini, L. T. Azzahro, R. A. Yuniar, and I. Prayogo, "Pengembangan Alat Bantu Pembelajaran Sorting Algorithm Berbasis Visual Console C++," *J. Ris. dan Apl. Mhs. Inform.*, vol. 06, no. 03, pp. 685–694, 2025.
[25] M. Marcellino and K. Margi, "Comparative of Advanced Sorting Algorithms (Quick Sort , Heap Sort , Merge Sort , Intro Sort , Radix Sort) Based on Time and Memory Usage," *2021 1st Int. Conf. Comput. Sci. Artif. Intell.*, vol. 1, no. October, pp. 154–160, 2021.
[26] A. H. Elkahlout and A. Y. A. Maghari, "A comparative Study of Sorting Algorithms Comb , Cocktail and Counting Sorting," *Int. Res. J. Eng. Technol.*, vol. 4, no. 1, pp. 1387–1390, 2017.
[27] V. Jugé, "Adaptive Shivers Sort : An Alternative Sorting Algorithm," *ACM Digit. Libr.*, vol. 20, no. 4, pp. 1–56, 2026, doi: 10.1145/3664195.
[28] E. Tan and D. Savin, "A New Class of Leonardo Hybrid Numbers and Some Remarks on Leonardo Quaternions over Finite Fields," *Mathematics*, vol. 2, no. 1, pp. 1–14, 2023.
[29] P. Rohith, S. Datta, C. D. Kamath, N. G. Kini, and A. R. B, "Parallelization of Pigeonhole Sort for Efficient Data Sorting," *Grenze Int. J. Eng. Technol.*, vol. 1, no. 6, pp. 6017–6021, 2024.
[30] H. Peters, O. Schulz-hildebrandt, N. Luttenberger, and A. B. Sort, "A novel sorting algorithm for many-core architectures based on adaptive bitonic sort," *IEEE 26th Int. Parallel Distrib. Process. Symp. A*, vol. 1, no. 227–237, 2012.
[31] M. Hasan, A. Rahman, K. Rahman, M. S. Rahman, M. Sharmin, and R. Yeasmin, "Pancake flipping and sorting permutations ☆," *J. Discret. Algorithms*, vol. 33, no. 1, pp. 139–149, 2015, doi: 10.1016/j.jda.2015.03.007.
[32] J. A. Caicedo, M. A. Uman, and J. T. Pilkey, "Lightning Evolution In Two North Central Florida Summer Multicell Storms and Three Winter / Spring Frontal Storms," *J. Geophys. Res. Atmos.*, vol. 1, no. 1, pp. 1155–1178, 2018.
[33] I. P. Pujiono, E. H. Rachmawanto, N. Anisa, and S. Winarsih, "Array Sorting Algorithm vs Algoritma Pengurutan Tradisional : Analisis Efisiensi Memori dan Waktu," *J. Manaj. Inform.*, vol. 15, no. April, pp. 47–59, 2025.
[34] Rifki, M. Faridz, T. S. Hidayah, and Suharsono, "Perbandingan Algoritma Selection Sort, Shell Sort, dan Merge Sort pada Data Sampling Numerik Menggunakan Matplotlib," *Pros. Semin. Nas. Sains dan Teknol. Seri 02*, vol. 1, no. 2, pp. 253–265, 2024.
[35] D. S. Sutanto, C. Kirana, and D. Wahyuningsih, "Analisis Efisiensi Waktu Bubble , Insertion , Merge , dan Quick Sort Menggunakan Python," *METIK J.*, vol. 9, no. 1, pp. 159–169, 2025.
[36] A. Arrosyidi and D. A. Arnandy, "Perbandingan Algoritma Simple Sorting antara Penggunaan Variabel Temporary dan Tanpa Variabel Temporary," *J. Sistim Inf. dan Teknol.*, vol. 4, no. 4, pp. 198–203, 2022, doi: 10.37034/jsisfotek.v4i2.185.
[37] A. S. Sabah *et al.*, "Comparative Analysis of the Performance of Popular Sorting Algorithms on

Datasets of Different Sizes and Characteristics," *Int. J. Acad. Eng. Res.*, vol. 7, no. 6, pp. 76–84, 2023.

[38] M. Mohammadagha, "Hybridization and Optimization Modeling, Analysis, and Comparative Study of Sorting Algorithms: Adaptive Techniques, Parallelization, for Mergesort, Heapsort, Quicksort, Insertion Sort, Selection Sort, and Bubble Sort," *Eng. Arch.*, vol. 1, no. 1, pp. 1–23, 2025.

[39] D. Setyantoro and R. A. Hasibuan, "Analisis dan Perbandingan Kompleksitas Algoritma Exchange Sort dan Insertion Sort untuk Pengurutan Data Menggunakan Python," *J. Ilm. Tek. Inform.*, vol. 21, no. 1, pp. 48–56, 2020.

[40] M. Z. Musyaffa, K. Raharjo, M. Faiz, S. Ammarullah, and Imam Prayogo Pujiono, "Efisiensi Memori dan Waktu: Array Sorting Algorithm vs Algortima Pengurutan Tradisional Menggunakan Python," *J. ELEKTRO Inform. SWADHARMA*, vol. 05, no. 02, pp. 72–81, 2025.

[41] A. Isyqi, G. D. Astuti, A. D. Saputro, M. A. Barryananda, and Suharsono, "Perbandingan Kinerja Algoritma Bubble Sort, Insertion Sort, dan Selection Sort Menggunakan Data Bilangan Numerik pada Program Python," *Pros. Semin. Nas. Sains dan Teknol. Seri 02*, vol. 1, no. 2, pp. 266–271, 2024.

[42] I. P. Pujiono *et al.*, "Algoritma Counting Sort vs Algoritma Pengurutan Modern: Analisis Memori dan Waktu Komputasi," *JITET (Jurnal Inform. dan Tek. Elektro Ter.*, vol. 13, no. 3, pp. 135–142, 2025.

[43] E. Taiwo, A. Oluwakemi, and A. Noah, "Comparative Study of Two Divide and Conquer Sorting Algorithms: Quicksort and Mergesort," *Procedia Comput. Sci.*, vol. 171, no. 2019, pp. 2532–2540, 2020.

[44] I. P. Pujiono, R. B. Trianto, and F. M. Hana, "Perbandingan Efisiensi Memori Dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java," *J. Sist. Inf. dan Sist. Komput.*, vol. 9, no. 2, pp. 218–230, 2024.

[45] N. Sari, W. A. Gunawan, P. K. Sari, I. Zikri, and A. Syahputra, "Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Menggunakan Bahasa Pemrograman Java," *ADI Bisnis Digit. Interdisiplin (ABDI Jurnal)*, vol. 3, no. 1, pp. 15–23, 2022.

[46] Fenina Adline Twince Tobing and J. R. Tambunan, "Analisis Perbandingan Efisiensi Algoritma Brute Force dan Divide and Conquer dalam Proses Pengurutan Angka," *ULTIMATICS*, vol. 12, no. 1, pp. 52–58, 2020.

[47] R. R. Basir, "Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge Dengan Pemrograman Java," *STRING (Satuan Tulisan Ris. dan Inov. Teknol.*, vol. 5, no. 2, pp. 109–118, 2020.

[48] M. L. Zulfa, Mikhael, and Betha Nurina Sari, "Analisis Perbandingan Algoritma Bubble Sort, Shell Sort, dan Quick Sort dalam Mengurutkan Baris Angka Acak menggunakan Bahasa Java," *J. Ilm. Wahana Pendidik.*, vol. 8, no. 13, pp. 237–246, 2022.

[49] M. Rai, Hemlata Parmar, and Suhani Sharma, "Comparative Analysis of Sorting Algorithms: Time Complexity and Practical Applications," *2025 3rd Int. Conf. Intell. Data Commun. Technol. Internet Things*, vol. 3, no. 1, pp. 26–32, 2025, doi: 10.1109/IDCIOT64235.2025.10915063.

[50] S. Zahwa, N. D. Amelia, R. Nafila, and R. A. Putri, "Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++," *J. Restikom Ris. Tek. Inform. dan Komput.*, vol. 7, no. 1, pp. 123–136, 2025.

[51] M. I. Ali, R. D. Fardiarsyah, L. Shodik, and F. Z. Dwi, "Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C ++," *LogicLink J. Artif. Intell. Multimed. Informatics*, vol. 2, no. 1, pp. 1–17, 2025.

[52] R. Latifah, E. Arriyanti, and A. R. Hakim, "Perbandingan Efisiensi Kinerja Algoritma Bubble Sort dan Algoritma Selection Sort pada Parallel Programming," *STMIK Widya Cipta Dharma (WICIDA)*, vol. 1, no. 1, pp. 1–7, 2021.

[53] Y. A. Astuti, "Analisis Pengujian Data Algoritma Bubble Sort," *Remik Ris. dan E-Jurnal Manaj. Inform. Komput.*, vol. 7, no. 3, pp. 1413–1420, 2023.

[54] A. Fenyi, M. Fosu, and Bright Appiah, "Comparative Analysis of Comparison and Non Comparison based Sorting Algorithms," *Int. J. Comput. Appl. (0975 – 8887)*, vol. 175, no. 28, pp. 22–25, 2020.

[55] M. A. 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," *J. Tek. Inform.*, vol. 9, no. 1, pp. 13–20, 2017.

[56] S. Wijaya, Fauziah, and T. W. Harjanti, "Perbandingan Algoritma Sorting Dengan Menggunakan Bahasa Pemograman Javascript Dalam Penggunaan Waktu Komputasi Dan Penggunaan Memori," *STRING (Satuan Tulisan Ris. dan Inov. Teknol.*, vol. 8, no. 3, pp. 294–302, 2024.

[57] M. N. Musyaafa, A. B. Simbolon, K. R. Azis, and D. Y. Niska, "Pengembangan Aplikasi Benchmarking Waktu Komputasi Algoritma Sorting Menggunakan Octave," *JATI (Jurnal Mhs. Tek. Inform.*, vol. 9, no. 4, pp. 6196–6200, 2025.