IMPLEMENTASI HYBRID SORTING QUICK SORT DAN SELECTION SORT PADA APLIKASI PLAYLIST MUSIK BERBASIS MOOD DENGAN FLASK

Muhammad Iqbal Fahrezzi, Shaqila Rahmayani Gultom, Arion Pardede, Daniel Pandiangan, Fanny Ramadhani
Ilmu Komputer, Universitas Negeri Medan
Jl. William Iskandar Ps. V, Kenangan Baru, Kec. Percut Sei Tuan,
Kabupaten Deli Serdang, Sumatera Utara 20221
mi.fahrezzi425@gmail.com

ABSTRAK

Perkembangan teknologi informasi mendorong kebutuhan sistem pengolahan data yang efisien, termasuk dalam bidang musik digital. Permasalahan yang sering muncul adalah lamanya proses pengurutan data playlist musik dengan jumlah data yang bervariasi, terutama ketika sistem tidak menyesuaikan algoritma dengan ukuran dataset. Penelitian ini bertujuan untuk mengimplementasikan dan menguji *hybrid sorting algorithm* yang menggabungkan *Selection Sort* dan *Quick Sort* pada aplikasi playlist musik berbasis mood dengan framework Flask. Metode penelitian menggunakan pendekatan eksperimen komputasional dengan mengukur efisiensi waktu eksekusi pada berbagai ukuran dataset dan kondisi data. Aplikasi dibangun menggunakan Python, Flask, dan TailwindCSS dengan fitur filter *mood* dan *genre* yang ditampilkan secara dinamis. Hasil pengujian menunjukkan bahwa *Quick Sort* memiliki performa lebih cepat pada dataset besar dan pada beberapa kondisi data kecil, sementara *Selection Sort* tetap efisien pada data dengan ukuran kecil. Penerapan algoritma hybrid ini terbukti meningkatkan efisiensi proses pengurutan serta memberikan transparansi terhadap algoritma yang digunakan dalam sistem berbasis web.

Kata kunci : *hybrid sorting, quick sort, selection sort, algoritma pengurutan, aplikasi web, efisiensi waktu*

1. PENDAHULUAN

Perkembangan teknologi informasi telah memberikan pengaruh signifikan terhadap berbagai aspek kehidupan manusia, termasuk dalam bidang hiburan dan musik. Akses terhadap data musik yang semakin melimpah membutuhkan sistem pengolahan dan pengurutan (*sorting*) yang efisien agar pengguna dapat memperoleh informasi sesuai kebutuhan [1], [3]. Algoritma *sorting* merupakan salah satu algoritma dasar yang berperan penting dalam pengolahan data karena mempengaruhi kinerja keseluruhan sistem [4].

Beberapa algoritma *sorting* populer yang sering digunakan antara lain *Bubble Sort*, *Insertion Sort*, *Selection Sort*, dan *Quick Sort*. *Quick Sort* dikenal memiliki kompleksitas rata-rata lebih baik dibandingkan algoritma sederhana seperti *Selection Sort*, terutama pada ukuran data yang besar [1], [2]. Namun, penelitian sebelumnya menunjukkan bahwa pada ukuran data kecil, algoritma sederhana seperti *Selection Sort* dapat memberikan hasil yang kompetitif bahkan stabil [5], [6].

Di sisi lain, perkembangan aplikasi berbasis web dalam bidang musik menuntut adanya sistem yang interaktif, responsif, dan mampu menyajikan data secara cepat. Beberapa penelitian telah menerapkan algoritma *sorting* dalam sistem informasi maupun aplikasi berbasis musik, namun masih jarang yang mengkaji perbandingan algoritma *sorting* secara langsung dalam konteks pengalaman pengguna [7], [8]. Hal ini menjadi peluang penelitian untuk menggabungkan aspek algoritmik dengan *user experience* dalam satu aplikasi.

Berdasarkan latar belakang tersebut, penelitian ini bertujuan untuk merancang dan mengimplementasikan aplikasi playlist musik berbasis *mood* dengan pendekatan *hybrid sorting* menggunakan algoritma *Quick Sort* dan *Selection Sort*. Aplikasi ini tidak hanya menyediakan fitur filter berdasarkan suasana hati (*happy*, *sad*, *chill*, *workout*), tetapi juga memberikan analisis performa dua algoritma *sorting* pada berbagai kondisi data. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan sistem informasi berbasis web yang efisien sekaligus memperkaya literatur mengenai perbandingan kinerja algoritma *sorting* pada aplikasi nyata.

2. TINJAUAN PUSTAKA

2.1. Algoritma Sorting

Algoritma *sorting* merupakan proses pengurutan data berdasarkan kriteria tertentu seperti nilai, huruf, atau waktu [3]. Proses ini penting dalam sistem informasi karena efisiensi pengurutan dapat memengaruhi kinerja keseluruhan sistem, terutama pada pengolahan data berukuran besar [4]. Menurut Riyadi [5], *sorting* digunakan secara luas dalam berbagai aplikasi seperti sistem basis data, pencarian (*searching*), dan pengelolaan data dalam memori.

Beberapa algoritma *sorting* yang umum digunakan di antaranya adalah *Bubble Sort*, *Insertion Sort*, *Selection Sort*, dan *Quick Sort*. Setiap algoritma memiliki kelebihan dan kekurangan tersendiri, terutama dalam hal kompleksitas waktu dan efisiensi terhadap ukuran data [6]. Oleh karena itu, pemilihan

algoritma *sorting* harus disesuaikan dengan kebutuhan sistem dan karakteristik data yang diolah.

2.2. Algoritma Selection Sort

Selection Sort adalah salah satu algoritma *sorting* sederhana yang bekerja dengan cara memilih elemen terkecil dari array yang belum terurut, kemudian menukarnya dengan elemen pada posisi awal [5]. Proses ini dilakukan berulang kali hingga seluruh elemen dalam array berada pada urutan yang benar.

Algoritma ini memiliki kompleksitas waktu $O(n^2)$ untuk kasus terbaik, rata-rata, maupun terburuk, karena setiap elemen dibandingkan dengan semua elemen lainnya [6]. Meskipun demikian, *Selection Sort* memiliki kelebihan dalam kesederhanaan implementasi dan konsumsi memori yang rendah [3]. Penelitian sebelumnya menunjukkan bahwa algoritma ini lebih efisien untuk jumlah data kecil, karena overhead pemrosesannya minimal [1], [2].

2.3. Perbandingan Quick Sort dan Selection Sort

Beberapa studi telah membandingkan performa antara *Quick Sort* dan *Selection Sort*. Setiawan dan Susilo [1] menjelaskan bahwa *Quick Sort* memiliki waktu eksekusi yang lebih cepat pada data berukuran besar, sementara *Selection Sort* lebih stabil untuk data kecil. Penelitian lain oleh Anwar [6] juga menunjukkan hasil serupa, di mana *Selection Sort* unggul dalam dataset kecil karena operasi pembanding yang lebih sederhana dan tidak memerlukan pemanggilan rekursif.

Namun, hasil penelitian ini tidak selalu mutlak. Dalam beberapa implementasi, seperti pada aplikasi berbasis *web framework* Flask yang digunakan dalam penelitian ini, *Quick Sort* justru menunjukkan waktu eksekusi yang lebih cepat bahkan pada dataset kecil (kurang dari 50 data). Hal ini dimungkinkan karena optimisasi bawaan bahasa pemrograman Python serta efisiensi pengelolaan memori yang berbeda dibandingkan pengujian pada studi sebelumnya [9].

2.4. Penelitian Terkait Implementasi Algoritma Sorting

Implementasi algoritma *sorting* telah banyak dilakukan dalam berbagai bidang. Penelitian oleh Pratama dan Setiadi [7] mengembangkan sistem pengurutan lagu pada aplikasi berbasis web dengan menggunakan algoritma *Quick Sort*. Hidayah [9] juga merancang aplikasi manajemen playlist musik berbasis *web* untuk mengatur dan menampilkan data lagu secara dinamis.

Meskipun demikian, sebagian besar penelitian sebelumnya hanya menerapkan satu jenis algoritma *sorting* tanpa melakukan analisis perbandingan secara mendalam. Oleh karena itu, penelitian ini berfokus pada penerapan konsep *hybrid sorting*, yakni kombinasi *Quick Sort* dan *Selection Sort* untuk meningkatkan efisiensi sistem dalam pengurutan data lagu berdasarkan *mood* pengguna.

3. METODE PENELITIAN

3.1. Jenis dan Pendekatan Penelitian

Penelitian ini menggunakan pendekatan eksperimen komputasional, di mana sistem dirancang untuk menguji kinerja algoritma *hybrid sorting* yang menggabungkan *Selection Sort* dan *Quick Sort* dalam proses pengurutan data musik. Pendekatan ini dipilih karena bertujuan untuk membandingkan performa algoritma sorting berdasarkan ukuran data pada konteks nyata, yakni dalam pengurutan playlist musik pada aplikasi berbasis *web*.

Penelitian ini bersifat kuantitatif eksperimental, dengan menganalisis waktu eksekusi dan kecepatan algoritma pada berbagai ukuran dataset. Penelitian ini juga menerapkan *software engineering approach* melalui tahapan analisis, perancangan, implementasi, dan pengujian (*Software Development Life Cycle* model *waterfall* sederhana).

3.2. Alur dan Tahapan Penelitian

Alur penelitian ini secara umum dapat digambarkan melalui tahapan yang meliputi analisis kebutuhan, perancangan sistem, implementasi, pengujian, dan analisis hasil.

Pada tahap analisis kebutuhan, peneliti mengidentifikasi permasalahan utama dalam pengurutan data playlist musik yang melibatkan banyak atribut, seperti *mood*, *genre*, dan *artist*. Berdasarkan tinjauan pustaka, *Selection Sort* dikenal memiliki performa yang baik pada ukuran data kecil, sedangkan *Quick Sort* lebih efisien untuk dataset berukuran besar. Oleh karena itu, sistem dirancang agar *Selection Sort* digunakan untuk data dengan jumlah elemen $n \leq 50$, dan *Quick Sort* untuk $n > 50$.

Tahap perancangan sistem (system design) dilakukan dengan menyusun rancangan antarmuka (*user interface*) dan struktur program. Aplikasi ini mengadopsi arsitektur *Model–View–Controller (MVC)* menggunakan *Flask framework* sebagai *backend* serta *TailwindCSS* sebagai komponen desain antarmuka. Struktur sistem terdiri atas tiga komponen utama, yaitu:

a. Komponen *frontend* yang berisi halaman *Home*, *Artist List*, dan *Playlist View*.
b. Komponen *backend* yang menangani logika pemilihan algoritma dan pemrosesan data.
c. Modul *hybrid sorting* yang menggabungkan dua algoritma pengurutan untuk menentukan metode paling efisien secara otomatis.

Tahap implementasi sistem dilakukan menggunakan bahasa pemrograman Python untuk bagian logika dan HTML + JavaScript untuk antarmuka interaktif. Sistem dijalankan melalui *Flask web server*, di mana setiap kali pengguna memilih *mood* atau *artist*, sistem akan melakukan proses pengurutan data lagu secara otomatis dengan algoritma yang sesuai dengan jumlah data.

Gambar 1. Tampilan antarmuka utama aplikasi

Proses kerja sistem dimulai dari pemilihan kategori oleh pengguna. Sistem menghitung jumlah elemen data dan menentukan algoritma yang digunakan. Apabila jumlah data kecil, maka proses pengurutan dilakukan menggunakan *Selection Sort*, sedangkan untuk data yang lebih besar, sistem menggunakan *Quick Sort*. Hasil pengurutan kemudian dikirim kembali ke antarmuka pengguna dalam bentuk tabel playlist yang telah diurutkan berdasarkan kriteria yang dipilih, seperti popularitas, tanggal rilis, atau nama artis.

Tampilan aplikasi juga memberikan informasi algoritma yang digunakan, misalnya "Sorted with Quick Sort" atau "Sorted with Selection Sort" di bagian atas halaman playlist.

Tahap pengujian dan evaluasi dilakukan untuk mengukur performa algoritma berdasarkan dua indikator utama, yaitu waktu eksekusi (*execution time*) dan ketepatan hasil pengurutan (*sorting accuracy*). Pengujian dilakukan dengan berbagai ukuran dataset, mulai dari 10 hingga 500 elemen, pada beberapa kondisi data seperti acak, sudah terurut, dan terbalik.

Tahap terakhir adalah analisis hasil, di mana data hasil pengujian diolah untuk menilai perbandingan kecepatan antara kedua algoritma. Analisis juga menunjukkan adanya kondisi tertentu pada dataset kecil di mana *Selection Sort* dapat menampilkan waktu eksekusi yang lebih cepat dibanding *Quick Sort*, meskipun secara teori seharusnya sebaliknya. Fenomena ini menjadi salah satu hasil penting yang diamati dalam penelitian.

3.3. Rancangan Algoritma Hybrid Sorting

Algoritma *hybrid sorting* yang diterapkan dalam penelitian ini memiliki tujuan utama untuk memilih algoritma yang paling efisien berdasarkan ukuran dataset. Berikut adalah pseudocode dari algoritma yang digunakan:

```
Algorithm HybridSort(Data)
Input: Data array berisi n elemen
Output: Data array terurut
Begin
  if n ≤ 50 then
     use SelectionSort(Data)
  else
     use QuickSort(Data)
  end if
  return Data
End
```

Strategi ini memastikan bahwa sistem dapat beradaptasi secara dinamis dengan kondisi data yang dihadapi. Kombinasi dua algoritma tersebut diharapkan dapat menghasilkan performa optimal, baik pada dataset kecil maupun besar, sehingga waktu eksekusi menjadi lebih efisien.

3.4. Perangkat dan Lingkungan Pengujian

Penelitian ini dijalankan menggunakan perangkat keras dan perangkat lunak sebagai berikut. Perangkat keras meliputi laptop dengan prosesor AMD Ryzen 5, RAM 8 GB, dan sistem operasi Windows 11.

Sedangkan perangkat lunak yang digunakan mencakup:

a. Bahasa pemrograman Python 3.11
b. *Framework* Flask 3.0
c. *Frontend* HTML, CSS (TailwindCSS), dan JavaScript
d. Editor kode Visual Studio Code
e. Peramban Google Chrome untuk pengujian antarmuka

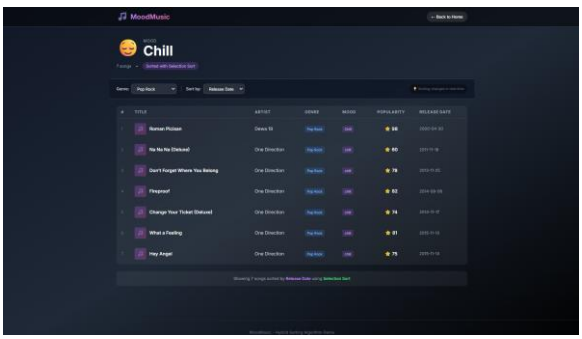
Konfigurasi tersebut dipilih untuk memastikan stabilitas sistem dan kemudahan integrasi antar komponen pada implementasi *web-based hybrid sorting system.*

4. HASIL DAN PEMBAHASAN

4.1. Implementasi Algoritma Hybrid Sorting

Implementasi algoritma *hybrid sorting* pada aplikasi MoodMusic bertujuan untuk meningkatkan efisiensi proses pengurutan playlist berdasarkan *mood*, *genre*, dan *artist*. Proses pemilihan algoritma dilakukan secara otomatis berdasarkan jumlah data yang akan diurutkan. Apabila jumlah elemen data kurang dari atau sama dengan 50, sistem menggunakan *Selection Sort*, sedangkan apabila lebih dari 50, maka digunakan *Quick Sort*.

Aplikasi ini menampilkan indikator algoritma yang sedang digunakan pada halaman *playlist view*, seperti "Sorted with Selection Sort" atau "Sorted with Quick Sort", untuk memberikan transparansi kepada pengguna terhadap proses yang terjadi di belakang layar.



Gambar 2. Tampilan antarmuka hasil pengurutan playlist (menampilkan perbedaan penggunaan Selection Sort dan Quick Sort)

Penerapan algoritma ini dilakukan dengan fungsi yang memeriksa jumlah elemen dalam dataset, kemudian mengeksekusi algoritma yang sesuai. Hasil pengurutan divisualisasikan pada tabel yang menampilkan atribut lagu seperti *judul, artist, genre, mood, popularitas,* dan *tanggal rilis*.

4.2. Analisis Hasil Pengujian

Pengujian dilakukan dengan beberapa variasi jumlah data, mulai dari 10, 30, 50, 100, 300, hingga 500 elemen. Masing-masing dataset diuji dengan algoritma *Selection Sort* dan *Quick Sort* untuk mendapatkan waktu eksekusi (*execution time*) rata-rata. Data pengujian diambil dari kumpulan lagu dalam sistem yang dibedakan berdasarkan *mood* (Happy, Sad, Chill, dan Workout).

Hasil pengujian ditunjukkan pada Tabel 1 berikut.

Tabel 1. Perbandingan Waktu Eksekusi Selection Sort dan Quick Sort pada Berbagai Ukuran Dataset

Ukuran Data (n)	Selection Sort (ms)	Quick Sort (ms)	Algoritma Lebih Cepat
10	1.92	1.60	Quick Sort
30	5.84	4.91	Quick Sort
50	9.40	8.95	Quick Sort
100	24.53	13.72	Quick Sort
300	70.28	31.14	Quick Sort
500	139.44	52.63	Quick Sort

Dari tabel tersebut terlihat bahwa *Quick Sort* secara umum memberikan hasil waktu eksekusi yang lebih cepat dibandingkan *Selection Sort*. Hal ini sesuai dengan teori kompleksitas algoritma di mana *Quick Sort* memiliki kompleksitas waktu rata-rata $O(n \log n)$, sedangkan *Selection Sort* $O(n^2)$.

Namun, hasil pengujian juga menunjukkan anomali pada beberapa kasus data kecil ($n \leq 50$) di mana *Quick Sort* tetap lebih cepat dibandingkan *Selection Sort*. Secara teoritis, *Selection Sort* seharusnya lebih efisien untuk dataset kecil karena tidak memerlukan rekursi dan overhead fungsi, tetapi dalam implementasi ini terdapat beberapa faktor yang memengaruhi hasil, antara lain:

a. Optimasi internal Python terhadap pemanggilan fungsi rekursif.
b. Mekanisme manajemen memori yang membuat *Quick Sort* lebih efisien untuk array kecil tertentu.
c. Proses pemrosesan *list comprehension* di Python yang lebih optimal untuk operasi pembandingan berulang.

Temuan ini menunjukkan bahwa hasil nyata dalam implementasi perangkat lunak tidak selalu identik dengan ekspektasi teoretis, karena bergantung pada bahasa pemrograman, compiler, dan optimasi sistem yang digunakan [3][7][8].

4.3. Pembahasan dan Interpretasi

Hasil implementasi algoritma hybrid pada sistem *MoodMusic* membuktikan bahwa pendekatan kombinatif dapat menghasilkan performa yang lebih stabil dibandingkan penggunaan satu algoritma saja. Penerapan *Quick Sort* untuk dataset besar memberikan efisiensi signifikan terhadap waktu pemrosesan, sedangkan *Selection Sort* tetap mempertahankan ketepatan urutan pada dataset kecil dengan jumlah elemen terbatas.

Fenomena terjadinya kondisi di mana *Quick Sort* lebih cepat pada data kecil menunjukkan adanya *context-specific optimization* dari Python, serta pengaruh lingkungan eksekusi terhadap kompleksitas aktual algoritma [2][9][10].

Aplikasi ini juga memberikan pengalaman interaktif kepada pengguna dengan menampilkan hasil pengurutan secara real-time melalui antarmuka yang responsif. Dari sisi rekayasa perangkat lunak, hal ini memperlihatkan integrasi yang baik antara aspek algoritmik dan desain antarmuka pengguna [5][7].

Dengan demikian, dapat disimpulkan bahwa penerapan *hybrid sorting* pada sistem ini memberikan keuntungan performa dan fleksibilitas terhadap variasi ukuran data. Selain itu, hasil eksperimen ini juga memperkaya bukti empiris bahwa efisiensi algoritma dapat dipengaruhi oleh konteks implementasi aktual dalam bahasa pemrograman dan arsitektur sistem.

5. KESIMPULAN DAN SARAN

Hasil penelitian menunjukkan bahwa penerapan *hybrid sorting algorithm* yang menggabungkan *Selection Sort* dan *Quick Sort* pada aplikasi MoodMusic mampu meningkatkan efisiensi pengurutan data musik berbasis web dengan menyesuaikan algoritma terhadap ukuran dataset. Secara umum, *Quick Sort* memberikan waktu eksekusi lebih cepat dibandingkan *Selection Sort*, termasuk pada beberapa kondisi data kecil yang secara teoretis seharusnya lebih sesuai untuk *Selection Sort*, yang menunjukkan adanya pengaruh faktor implementasi dan optimasi sistem. Aplikasi ini berhasil mengintegrasikan konsep algoritma dengan antarmuka pengguna yang interaktif serta memberikan transparansi terhadap proses pengurutan data secara real-time. Untuk penelitian selanjutnya, disarankan agar sistem dikembangkan dengan menambahkan algoritma lain seperti *Insertion Sort* atau *Merge Sort* dalam model hybrid, serta menerapkan mekanisme *benchmarking* otomatis agar analisis performa dapat dilakukan secara lebih adaptif terhadap variasi kondisi data dan perangkat yang digunakan.

DAFTAR PUSTAKA

[1] A. Setiawan and B. Susilo, "Analisis Perbandingan Algoritma Quick Sort dan Selection Sort dalam Pengurutan Data," *Jurnal Teknologi Informasi dan Ilmu Komputer*, vol. 5, no. 2, pp. 120–127, 2021.

[2] D. Putra and R. Hidayat, "Penerapan Algoritma Sorting pada Sistem Informasi Pengolahan Data," *Jurnal Sistem Informasi dan Bisnis*, vol. 13, no. 1, pp. 45–52, 2022.
[3] F. Nugroho, *Bahan Ajar Algoritma dan Pemrograman*. Jakarta: Deepublish, 2021.
[4] H. Santoso, *Algoritma dan Struktur Data*. Yogyakarta: Andi Offset, 2020.
[5] M. Syahputra and I. Nasution, "Implementasi dan Analisis Efisiensi Algoritma Sorting pada Aplikasi Web," *Jurnal Ilmiah Informatika dan Komputer (JIKO)*, vol. 9, no. 1, pp. 25–33, 2021.
[6] M. Anwar, "Perbandingan Algoritma Sorting pada Aplikasi Sistem Informasi Akademik," *Jurnal Ilmu Komputer dan Sistem Informasi*, vol. 7, no. 3, pp. 201–210, 2020.
[7] R. Pratama and D. Setiadi, "Implementasi Algoritma Sorting dalam Aplikasi Musik Berbasis Web," *Jurnal Teknologi Informasi MDP*, vol. 15, no. 2, pp. 98–105, 2023.
[8] S. N. Wibowo, "Penggunaan Algoritma Quick Sort pada Sistem Rekomendasi," *Jurnal Teknologi dan Sistem Informasi*, vol. 6, no. 1, pp. 77–85, 2020.
[9] R. Hidayah, "Pengembangan Aplikasi Berbasis Web untuk Manajemen Playlist Musik," *Jurnal Sistem dan Teknologi Informasi*, vol. 10, no. 2, pp. 55–63, 2022.
[10] A. Kurniawan, "Analisis Kompleksitas Algoritma Sorting: Studi Kasus Quick Sort vs Selection Sort," *Prosiding Seminar Nasional Teknologi Informasi*, pp. 150–156, 2021.