e-ISSN 2549-7472
Edumatic: Jurnal Pendidikan Informatika
Vol. 7 No. 1, Juni, 2023, **Hal. 39-48**
DOI: 10.29408/edumatic.v7i1.12358

Optimasi Pengurutan Data Bilangan dengan Menggabungkan Algoritma *Selection Sort Hybrid* dan *Bucket Sort*

Risqi Pradana Aryanto [1,*], Agung Nilogiri [1], Ari Eko Wardoyo [1]

[1] Program Studi Teknik Informatika, Universitas Muhammadiyah Jember, Indonesia
[*] Correspondence: riskipradana221001@gmail.com

Received: 27 Maret 2023 | Revised: 8 April 2023 | Accepted: 15 April 2023 | Published: 20 Juni 2023

Abstrak
Algoritma *sorting* sangat penting dalam pengolahan data, terutama untuk data bilangan bulat. Namun, semakin banyak bilangan yang diurutkan, semakin lama waktu yang dibutuhkan oleh algoritma *sorting*, terutama untuk algoritma *sorting* dengan kompleksitas $O(n^2)$. Artikel ini membahas tentang optimasi pengurutan data bilangan dengan menggabungkan algoritma *Selection Sort Hybrid* dan *Bucket Sort*. Penelitian ini bertujuan untuk menguji performa dari algoritma *Selection Sort Hybrid* dan *Bucket Sort* serta membandingkannya dengan algoritma pengurutan data lainnya. Metode penelitian yang digunakan adalah kuantitatif eksperimental dengan pengambilan data bilangan acak menggunakan bahasa pemrograman Python. Data tersebut diuji dengan algoritma pengurutan Gabungan *Selection sort hybrid* dengan *bucket sort*, *Selection Sort Hybrid*, *Quick Sort*, dan *Merge Sort*. Analisis data dilakukan dengan menghitung waktu eksekusi masing-masing algoritma pengurutan. Hasil penelitian menunjukkan bahwa algoritma *Selection Sort Hybrid* dan *Bucket Sort* lebih cepat daripada algoritma pengurutan lainnya dalam pengujian menggunakan data bilangan yang besar dan kompleks. Oleh karena itu, penggunaan kombinasi antara *Selection Sort Hybrid* dan *Bucket Sort* dapat meningkatkan efisiensi dan kecepatan dalam pengurutan data bilangan yang kompleks.

Kata kunci: algoritma *sorting*; *bucket sort*; optimasi; *selection sort hybrid*

Abstract
Sorting algorithms are crucial in data processing, particularly for integer data. However, as the number of integers to be sorted increases, the sorting algorithm takes longer to complete, especially for algorithms with O(n2) complexity. This article discusses optimizing integer data sorting by combining the Selection Sort Hybrid and Bucket Sort algorithms. The study aims to test the performance of the Selection Sort Hybrid and Bucket Sort algorithms and compare them with other data sorting algorithms. The research method used is experimental quantitative research using randomly generated data using Python. The data were tested using the Combined Selection Sort Hybrid with Bucket Sort algorithm, Selection Sort Hybrid, Quick Sort, and Merge Sort. Data analysis was done by calculating the execution time of each sorting algorithm. The results show that the Selection Sort Hybrid and Bucket Sort algorithms are faster than other sorting algorithms in testing with large and complex integer data. Therefore, combining Selection Sort Hybrid and Bucket Sort algorithms can improve the efficiency and speed of sorting complex integer data.

Keywords: *sorting algorithm; bucket sort; optimization; selection sort hybrid*

PENDAHULUAN

Pengurutan data atau *Sorting* merupakan suatu proses penyusunan kembali beberapa kumpulan data yang sebelumnya telah tersusun secara acak hingga tersusun secara teratur menurut aturan tertentu. Data ini biasanya bertipe numerik dan karakter. (Sari et al., 2019).

Dalam proses pengurutan data numerik pada komputer, Dibutuhkan suatu metode yang tepat karena bisa mempengaruhi hasil yang ingin di capai. Sebaik apapun algoritma, jika menghasilkan output yang salah, maka algoritma tersebut bukanlah algoritma yang baik.(Adline et al., 2020a) Algoritma *sorting* merupakan topik yang penting dalam ilmu komputer dan memiliki berbagai macam algoritma yang berbeda dengan kecepatan dan kompleksitas yang beragam(Arifin & Setiyadi, 2020). Pemilihan algoritma *sorting* yang tepat akan memberikan efisiensi dalam pengolahan data. (Puspita Sari et al.)

Namun, pada proses pengurutan data acak bilangan integer yang berjumlah besar terutama data bilangan integer dengan banyak data acak lebih dari 1.000 bilangan dapat menjadi lebih lambat terutama untuk algoritma *sorting* dengan kompleksitas $O(n^2)$. (Anggreani et al., 2020) Oleh karena itu, diperlukan algoritma *sorting* yang efisien dan lebih cepat untuk mengurutkan data bilangan acak integer menjadi urutan data bilangan interger yang benar dalam jumlah yang besar. Beberapa algoritma *sorting* seperti *Selection Sort*, Bubble *Sort*, dan Insertion *Sort* mempunyai kompleksitas yang relatif besar yaitu $O(n^2)$. (Chauhan & Duggal, 2020; Basir, 2020; Setyantoro & Hasibuan, 2020) sehingga kurang efektif dalam mengolah data yang besar. (Toyib et al., 2021) Sementara itu, algoritma *sorting* seperti *Quick Sort* mempunyai waktu eksekusi yang lebih baik dibandingkan algoritma *sorting* seperti bubble *sort* (Poetra, 2022) karena menerapkan teknik *devide and conquer* (Hasibuan, 2022; Tumanggor et al., 2022) yaitu dengan membagi deret menjadi sub-deret dan sub-deret tersebut dibagi menjadi beberapa sub-sub deret lagi sehingga membutuhkan waktu eksekusi yang cepat (Andri, 2019)

Mengatasi permasalahan tersebut, perlu untuk membuat atau menggabungkan suatu algoritma dengan algoritma atau metode lainnya agar mempunyai komplektisitas waktu yang lebih rendah (Hastomo et al.,2021.; Nishom & Fathoni, 2018) . Penggabungan antara dua algoritma *sorting* dengan menerapkan teknik *devide and conquer* bisa menurunkan komplektisitas waktu sehingga meningkatkan waktu eksekusi dari suatu algoritma karena pada teknik *devide and conquer* masalah di pecahkan menjadi beberapa sub-masalah kecil kemudian masalah tersebut diselesaikan pada setiap sub-masalah (Ilmu et al., 2021).

Algoritma seperti *Bucket Sort* dan *selection sort* memiliki kelebihan dan kekurangan masing-masing dalam menangani data yang berbeda. *Bucket Sort* merupakan algoritma pengurutan non-perbandingan yang menerapkan konsep membagi nilai dalam *array* menjadi diurutkan ke dalam ember/ *array* . Kemudian, setiap *bucket* / *array* tersebut akan diurutkan menggunakan algoritma *sorting* tertentu. Setelah itu, elemen yang sudah terurut dari setiap *bucket* akan digabungkan menjadi satu *array* hasil akhir. (Gill et al., 2019). Meskipun *bucket sort* sudah cukup efisien, namun algoritma *sorting* yang digunakan pada setiap *bucket* juga mempengaruhi performa dari algoritma *bucket sort* secara keseluruhan sehungga *bucket sort* lebih cocok digunakan untuk data yang tersebar di seluruh rentang nilai (Wei et al., 2021). Sementara itu *Selection Sort* adalah salah satu algoritma pengurutan yang bekerja dengan cara memilih elemen terkecil dari kumpulan data dan memindahkannya ke posisi yang sesuai (Sari et al., 2022) . Proses ini diulang untuk setiap elemen dalam kumpulan data hingga seluruh data terurut. *Selection sort* lebih efektif untuk mengurutkan data yang sudah hampir terurut karena *Selection Sort* memiliki komplektisitas $O(n^2)$ (Aqib et al., 2021). Selain itu optimasi terhadap suatu algoritma bisa meningkatkan komplektisitas dari algoritma itu sendiri (Ramli, et al. 2018). Salah satu optimasi algortima *selection sort* adalah *selection sort hybrid*.

Optimasi *Selection sort* (Hardika et al., 2020) hanya membahas mengenai optimasi kerja algoritma *selection sort* secara *multithreading* (*selection sort hybrid*) yaitu mengurutkan elemen-elemen dengan mencari nilai minimum dan maksimum, kemudian menukar posisi nilai minimum dengan nilai pada indeks awal, dan menukar posisi nilai maksimum dengan nilai pada indeks terakhir. dalam penelitian ini tidak mempertimbangkan untuk menggunakan strategi lainnya seperti menggabungkan algoritma dan teknik lainnya untuk meningkatkan efisiensi waktu. Dalam penelitian ini, komplektisitas algoritma *Selection sort hybrid* masih

cukup tinggi yaitu $O(n^2)$ sehingga walaupun algoritma *selection sort* sudah di optimasi, masih tetap memiliki komplektisitas yang tinggi sehingga algoritma *Selection sort hybrid* masih kurang efisien jika untuk mengurutkan data dalam jumlah yang banyak. Algoritma *bucket sort* memiliki komplektisitas waktu O(n log n). Hal ini disebabkan karena algoritma *bucket sort* menggunakan metode *devide and conquer* yaitu dengan membagi deret menjadi beberapa sub-*array* dan mengurutkan elemen pada setiap *array* secara terpisah menggunakan algoritma *sorting* lainnya dan menggabungkan kembali semua *array* yang terpisah menjadi satu deret yang terurut (Kumar et al., 2020). Sementara itu, penelitian ini akan membahas penggunaan *Selection sort hybrid* dengan metode *bucket sort* yang akan mengelompokkan elemen-elemen yang ada pada *array* ke dalam beberapa *range* atau *bucket*, dan kemudian melakukan *sorting* pada masing-masing *bucket* menggunakan algoritma *Selection sort hybrid*. Setelah itu, algoritma menggabungkan kembali semua *bucket* menjadi satu *array* yang terurut. Dari analisa ini, dapat disimpulkan bahwa penelitian sebelumnya lebih berfokus pada penggunaan algoritma *Sorting Selection sort hybrid* dalam pengurutan data secara umum, sementara penelitian ini lebih fokus pada pengembangan algoritma *Sorting Selection sort hybrid* dengan *Bucket Sort* sebagai solusi optimal dalam pengurutan data dalam jumlah yang besar dan kompleks.

Tujuan penelitian ini yaitu untuk mengoptimalkan algoritma pengurutan *Selection sort hybrid* dengan menggunakan *Bucket Sort*, sehingga mampu menghasilkan waktu eksekusi dalam pengurutan data yang lebih efektif dan efisien. Selain itu, penelitian ini juga bertujuan untuk mengevaluasi kinerja algoritma *selection sort hybrid* dengan *bucket sort* pada data dalam jumlah besar dan kompleks, serta membandingkannya dengan algoritma pengurutan *selection sort hybrid quick sort*, dan *merge sort*. Dengan demikian, penelitian ini diharapkan dapat memberikan kontribusi dalam pengembangan algoritma pengurutan data yang lebih efisien dan efektif dalam menangani data dalam jumlah besar dan kompleks.

METODE

Penelitian ini menggunakan metode penelitian kuantitatif eksperimental. Metode eksperimental digunakan untuk menguji hipotesa atau menjawab pertanyaan penelitian dengan mengontrol variabel-variabel yang mempengaruhi fenomena yang diteliti. Metode kuantitatif digunakan untuk mengukur kinerja algoritma secara objektif dengan menghasilkan data numerik yang dapat dianalisis menggunakan metode statistik. Dalam penelitian ini, variabel yang diuji adalah kinerja algoritma pengurutan data yang di ukur dari waktu eksekusi menggunakan kombinasi antara *Sorting Selection sort hybrid* dengan *Bucket Sort*, algoritma *Selection sort hybrid* dan algoritma *quick sort*.

Penelitian ini menggunakan metode pengujian kinerja menggunakan subjek bilangan numerik acak. Pada pengujian ini data bilangan numerik acak dihasilkan dengan menggunakan fungsi random pada bahasa pemrograman Python. Data dihasilkan dengan menggunakan beberapa ukuran *array* yang berbeda-beda, yaitu 1.000, 10.000, 100.000 , dan 1.000.000 data. Setiap data diacak sebelum disimpan dalam *array* untuk pengujian. Selain itu, penelitian ini juga menggunakan teknik pengukuran waktu eksekusi yang di ukur dengan library *time* pada program *python* untuk mengukur kinerja algoritma pengurutan data bilangan numerik acak. Teknik ini digunakan untuk mengukur waktu yang dibutuhkan oleh algoritma untuk menyelesaikan pengurutan data. Waktu eksekusi diukur menggunakan fungsi waktu pada program *Python*. Penelitian ini menggunakan pendekatan komparatif untuk menguji kinerja algoritma pengurutan data baru yang dikembangkan. Dalam pendekatan ini, algoritma pengurutan data baru akan dibandingkan dengan algoritma pengurutan data lainnya, yaitu *Selection sort hybrid* dan *quick sort*.

Desain eksperimen yang digunakan pada penelitian ini adalah *pre-test post-test* design. Dalam desain ini, dilakukan pengukuran kinerja algoritma pengurutan data sebelum dan setelah

dilakukan pengujian. Pengukuran dilakukan dengan cara membandingkan waktu eksekusi pada ketiga algoritma. Pengujian dilakukan dengan membagi data ke dalam beberapa ukuran *array* yang berbeda-beda Data yang dihasilkan dari setiap pengujian akan direkap dan dianalisis untuk mendapatkan hasil pengujian yang valid.

Tabel 1. Desain penelitian

Kelompok eksperimen	**Pretest**	**Waktu eksekusi**	**Posttest**
Selection sort hybrid	T_1	X_s	T_2
Selection sort hybrid dengan gabungan *bucket sort*	T_1	X_{sb}	T_2
Quick Sort	T_1	X_{sq}	T_2
Merge Sort	T_1	X_{sm}	T_2

Keterangan :

T_1 = *pretest* (*tes awal)*
T_2 = *posttest* (*tes akhir)*
X_s = Waktu eksekusi algoritma *Selection sort hybrid*
X_{sb} = Waktu eksekusi gabungan algoritma *selection hybrid* dengan *bucket sort*
X_{sq} = Waktu eksekusi algoritma *quick sort*
X_{sm} = Waktu eksekusi algoritma *merge sort*

Kelompok eksperimen pada tabel 1 merupakan subjek atau obyek yang menjadi fokus dalam penelitian. Terdapat empat kelompok eksperimen, yaitu kelompok yang menjalankan metode *Selection sort hybrid*, kelompok yang menjalankan metode *Selection sort hybrid* dengan gabungan *bucket sort*, kelompok yang menjalankan metode *Merge sort* dan kelompok yang menjalankan metode *Quick sort*. Pretest (t1) dan posttest (t2) pada tabel tersebut merupakan pengukuran kondisi awal dan akhir kelompok eksperimen dalam penelitian. Pretest dilakukan sebelum penelitian dimulai untuk mengetahui kondisi awal dari kelompok eksperimen, sedangkan posttest dilakukan setelah penelitian selesai untuk mengetahui hasil yang diperoleh setelah kelompok eksperimen menjalankan metode pengurutan yang ditetapkan. Waktu eksekusi (X) pada tabel tersebut merupakan waktu yang dibutuhkan oleh setiap kelompok eksperimen dalam mengeksekusi metode pengurutan. Hal ini penting untuk diketahui karena waktu eksekusi yang lebih cepat dapat menunjukkan efektivitas dan efisiensi suatu metode pengurutan.

Subjek penelitian dalam penelitian ini yaitu data numerik acak yang dihasilkan dengan menggunakan fungsi random pada bahasa pemrograman Python. Data dihasilkan dengan menggunakan beberapa ukuran *array* yang berbeda-beda, yaitu 1.000, 10.000, 100.000 dan 1.000.000 elemen. Setiap ukuran *array* diuji sebanyak 10 kali untuk memastikan hasil pengujian yang lebih akurat. Data yang dihasilkan akan digunakan untuk membandingkan kinerja algoritma *Bucket Sort* dengan algoritma *Selection sort hybrid* dengan *Selection sort hybrid* konvensional dalam melakukan pengurutan data. Data akan dianalisis untuk mengetahui perbedaan waktu eksekusi dan efisiensi pengurutan data pada kedua algoritma.

Teknik pengumpulan data pada penelitian ini dilakukan dengan membuat program Python yang menghasilkan data numerik acak dengan fungsi random pada bahasa pemrograman Python. Data kemudian akan disimpan pada sebuah file untuk dapat diproses oleh program pengujian algoritma. Teknik analisa yang digunakan dalam penelitian ini yaitu teknik analisis statistik deskriptif. Data waktu eksekusi dan jumlah langkah operasi dasar dari kedua algoritma akan dihitung dan dianalisis untuk mengetahui perbedaan kinerja algoritma

dalam pengurutan data. Selain itu, grafik waktu eksekusi dan jumlah operasi dasar juga akan dibuat untuk memvisualisasikan hasil pengujian secara lebih jelas.

HASIL DAN PEMBAHASAN

Hasil

Pada penelitian ini, digunakan desain penelitian *pre-test post-test* untuk mengevaluasi kinerja algoritma *sorting*. Pada tahap *pre-test*, dikumpulkan data dalam bentuk deret angka integer dengan ukuran *array* yang berbeda-beda yaitu 1.000, 10.000, 100.000, dan 1.000.000 elemen. Selanjutnya, diukur kinerja algoritma *sorting selection sort hybrid* dan algoritma *sorting* lainnya seperti *quick sort* pada setiap ukuran *array* yang telah ditentukan dengan membandingkan waktu eksekusi pada algoritma *sorting*. Setelah melakukan *pre-test*, dikembangkan algoritma baru dengan menggabungkan *selection sort hybrid* dan *bucket sort*. Algoritma baru ini kemudian diuji pada ukuran *array* yang sama seperti pada tahap *pre-test*. Pada tahap *post-test*, kinerja algoritma *sorting* kembali di ukur pada setiap ukuran *array* dengan menggunakan algoritma baru yang telah dikembangkan. Hasil pengujian yang diperoleh dari *pre-test* dan *post-test* akan direkap dan dianalisa untuk mendapatkan hasil pengujian yang valid. Analisis data akan digunakan untuk mengevaluasi kinerja waktu eksekusi algoritma *sorting* dan menentukan keunggulan algoritma baru yang telah dikembangkan. Dalam penelitian ini, digunakan tiga jenis algoritma *sorting* yang berbeda, yaitu *selection sort hybrid* , gabungan *selection sort hybrid* dengan *bucket sort* dan *quick sort*. Hal ini dilakukan untuk membandingkan kinerja kedua algoritma dan mengevaluasi keunggulan algoritma baru yang telah dikembangkan. Langkah yang terakhir yaitu mencatat dan membandingkan waktu eksekusi ke dua algoritma tersebut dengan membuat tabel perbandingan yaitu tabel 2 dan diilustrasi dalam bentuk grafik yang dapat dilihat pada gambar 1.

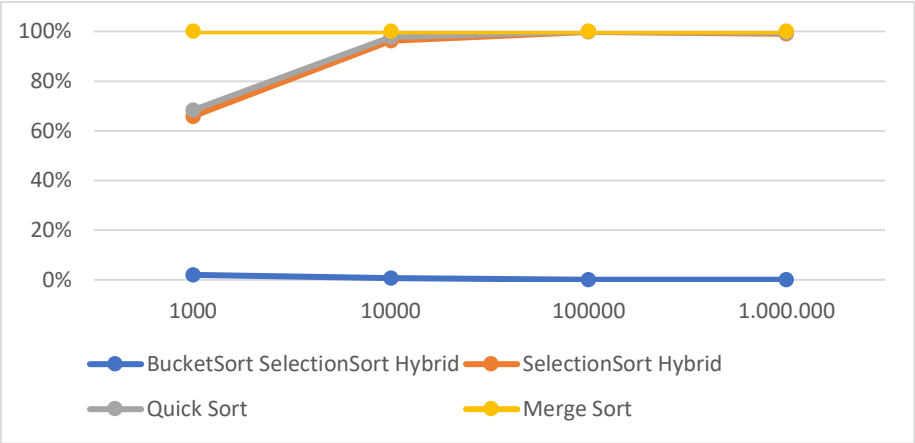
Tabel 2. Perbandingan waktu eksekusi algoritma *bucket sort* menggunakan *selection hybrid* dengan algoritma lainnya

Banyak Data	Waktu Eksekusi (s)			
	BucketSort SelectionSort Hybrid	***SelectionSort Hybrid***	***Quick Sort***	***Merge Sort***
1.000	0,001003	0,0324	0,00123	0,01611
10.000	0,013185	2,0538	0,03070	0,04908
100.000	0,109817	343,604	0,38066	0,43318
1.000.000	1,535877	1299,92	6,83016	6,35155

Berdasarkan data pada tabel 2, algoritma *Selection sort hybrid* dengan *bucket sort* menghasilkan waktu eksekusi yang lebih baik dari algoritma *Selection sort hybrid* dan *quick sort* karena memiliki kompleksitas waktu rata-rata yang lebih rendah.

Pada *Selection sort hybrid* dengan *bucket sort*, algoritma *Selection sort hybrid* digunakan pada setiap *bucket*, yang memiliki kompleksitas waktu terbaik $O(n^2)$ dan waktu terburuk $O(n^2)$, namun karena digunakan hanya pada ukuran subset yang kecil dari, maka kompleksitas waktu total algoritma tetap terjaga. Sedangkan pada algoritma *Selection sort hybrid*, meskipun menggunakan teknik pengurutan yang lebih efisien daripada *Selection Sort* konvensional yaitu dengan mengelompokkan nilai terkecil pada deret sebelah kiri dan terbesar pada sebelah kanan di bagian pengurutan, tetapi tetap menggunakan teknik *Selection Sort* pada pemilihan elemen terkecil dan terbesar dari semua deret tanpa dilakukan pembagian deret menjadi beberapa sub deret. Sehingga meskipun pengurutan dilakukan dengan lebih efisien, namun pemilihan elemen

terkecil dan terbesar tetap membutuhkan waktu yang cukup signifikan dan dapat mempengaruhi waktu eksekusi algoritma secara keseluruhan.



Gambar 1. Hasil perbandingan waktu eksekusi

Oleh karena itu, *Selection sort hybrid* dengan *bucket sort* memiliki kompleksitas waktu rata-rata yang lebih rendah dibandingkan dengan algoritma *Selection sort hybrid* sehingga algoritma ini lebih efisien untuk digunakan pada data numerik dengan rentang nilai yang lebih besar. *Pseudocode* algoritma gabungan antara *selection sort hybrid* dapat di lihat pada Gambar 2.

```
# (Langkah Pertama) Mencari nilai maksimum dan minimum dalam array
max_val = maximum(array)
min_val = minimum(array)

# (Langkah Kedua) Menentukan ukuran setiap range
range_size = 10
num_ranges = ((max_val - min_val) // range_size) + 1

# (Langkah Ketiga) Inisialisasi array kosong untuk setiap range
ranges = []
for i in range(num_ranges):
    ranges.append([])

# (Langkah Keempat) Memasukkan setiap elemen ke dalam range yang sesuai
for val in array:
    range_index = (val - min_val) // range_size
    ranges[range_index].append(val)

# (Langkah Kelima) Mengurutkan setiap range menggunakan selection sort hybrid
for r in ranges:
    n = len(r)
    for i in range(n - 1):
        min_idx = i
        max_idx = i
        for j in range(i + 1, n):
            if r[j] < r[min_idx]:
                min_idx = j
            if r[j] > r[max_idx]:
                max_idx = j
        if min_idx != i:
            r[i], r[min_idx] = r[min_idx], r[i]
        if max_idx == i:
            max_idx = min_idx
        if max_idx != n - 1:
            r[n - 1], r[max_idx] = r[max_idx], r[n - 1]

# (Langkah Keenam) Menggabungkan setiap range menjadi satu array hasil akhir
sorted_ranges = []
for r in ranges:
    sorted_ranges.extend(r)

return sorted_ranges
```

Gambar 2. *Pseudocode* algoritma

Pada impelentasi bentuk data bilangan integer, maka Langkah – Langkah kerja algoritma gabungan *bucket sort* dengan *selection hybrid* berdasarkan Gambar 2. yaitu, Langkah pertama adalah mencari nilai maksimum dan minimum dalam *array* dari suatu data yang akan diurutkan. Dalam contoh deret yang diberikan, yaitu *array* A = [43, 8, 56, 2, 76, 12, 90, 33, 21, 67, 98, 4, 31, 55, 78], nilai maksimum yaitu 98 dan nilai minimum yaitu 2. Setelah itu, pada Langkah ke dua menentukan ukuran setiap *range*. Ukuran *range* ditentukan oleh faktor-faktor seperti jumlah data dan rentang nilai yang ada pada data tersebut. Pada *Pseudocode* Gambar 2, setiap *range* menyimpan maksimal 10 deret dengan rentang dimulai dari nilai bilangan terkecil

yang ditemukan pada setiap deret. Pada *array* A nilai minimum nya 2 sehingga rentang *range* pertama dimulai dari nilai 2 dan seterusnya hingga menampung sebanyak 10 bilangan.

Setelah menentukan ukuran *range*, langkah ketiga adalah melakukan inisialisasi *array* kosong untuk setiap *range*. Langkah keempat yaitu memasukkan setiap elemen ke dalam *range* yang sesuai. Setiap elemen bilangan pada data dihitung berdasarkan *range* dimana data tersebut akan dimasukkan. Ini dilakukan dengan menghitung indeks *bucket* menggunakan rumus (val - min_val) // *range*_size. Setiap elemen kemudian dimasukkan ke dalam *bucket* yang sesuai. Pembangian setiap elemen ke dalam *range* dapat di lihat pada tabel 3. *Range*s 0 Merupakan kelompok bilangan yang dimulai dari bilangan terkecil yang ditemukan yaitu 2 dan menampung sebanyak maksimal 10 data bilangan sehingga *range*s 0 memiliki rentang 2 – 11, *range*s 1 memiliki rentang 12 – 22, *range*s 2 memiliki rentang 23 – 32, *range*s 3 memiliki rentang 33 – 42, *range*s 4 memiliki rentang 43 – 52, *range*s 5 memiliki rentang 53 – 62, *range*s 6 memiliki rentang 63 – 72, *range*s 7 memiliki rentang 72 – 81, *range*s 8 memiliki rentang 81 – 90.

Tabel 3. Pembagian setiap deret data integer ke setiap *range*

***Range*s ke -**	**Isi *Range*s**
*range*s[0]	[2, 4, 8]
*range*s[1]	[12, 21]
*range*s[2]	[31]
*range*s[3]	[33]
*range*s[4]	[43]
*range*s[5]	[55, 56]
*range*s[6]	[67]
*range*s[7]	[76, 78]
*range*s[8]	[90]

Pada langkah kelima, Bilangan dalam setiap *range* akan diurutkan menggunakan algoritma *Selection sort hybrid*. Pada setiap *range*, algoritma akan memilih elemen bilangan terkecil dan memindahkannya ke posisi pertama pada *range*. Selanjutnya, algoritma akan mencari elemen terkecil pada posisi kedua hingga akhir *range*, dan memindahkannya ke posisi kedua. Algoritma akan terus berlanjut hingga seluruh elemen pada *range* terurut. Pada langkah keenam, setiap *range* yang telah terurut akan digabungkan menjadi satu *array* hasil akhir. Dalam kasus ini, hasil akhir yang dihasilkan adalah *sort*ed_*range*s = [2, 4, 8, 12, 21, 31, 33, 43, 55, 56, 67, 76, 78, 90, 98]. *Array* ini merupakan *array* telah terurut dari terkecil hingga terbesar.

Pembahasan

Pada penelitian ini dilakukan penggunaan algoritma gabungan antara algoritma *bucket sort* dan *Selection sort hybrid* untuk mengurutkan *array*. Algoritma *bucket sort* membagi elemen-elemen *array* ke dalam beberapa *bucket* atau *range* tergantung pada nilai elemen tersebut dan menggunakan algoritma *Selection sort hybrid* untuk mengurutkan data pada setiap *array*. Langkah awal pada algoritma *bucket sort* adalah mencari nilai maksimum dan minimum dalam *array* untuk menentukan rentang nilai yang akan digunakan dalam pembagian elemen *array* ke dalam setiap *range*. Setelah itu, ukuran setiap *range* ditentukan berdasarkan selisih nilai maksimum dan minimum, kemudian dibagi dengan *range_size*. Dilakukan inisialisasi *array* kosong untuk setiap *range*, yang nantinya akan digunakan untuk menampung elemen-elemen yang masuk ke dalam *range* yang sesuai. Setiap elemen *array* kemudian dimasukkan ke dalam *range* yang sesuai dengan menghitung indeks *range* untuk setiap elemen *array* dan memasukkannya ke dalam *array* kosong pada *range* yang sesuai. Setelah itu, setiap *range* yang

telah berisi elemen akan diurutkan menggunakan *Selection sort hybrid*. Setelah setiap *range* telah diurutkan, langkah terakhir adalah menggabungkan setiap *range* yang telah diurutkan menjadi satu *array* hasil akhir yang sudah terurut.

Hasil penelitian ini menunjukkan bahwa penggunaan algoritma *selection sort hybrid* dengan *bucket sort* dapat mengurutkan data bilangan integer secara benar serta meningkatkan efisiensi pengurutan *array*. Hal ini disebabkan oleh adanya pengurutan pada setiap *bucket* dengan *Selection sort hybrid* yang telah dimodifikasi, yaitu dengan mengurutkan elemen-elemen dengan nilai minimum dan maksimum, kemudian menukar posisi nilai minimum dengan nilai pada indeks awal *range*, dan menukar posisi nilai maksimum dengan nilai pada indeks terakhir *range*. Proses pengurutan menjadi lebih cepat karena jumlah elemen pada setiap *bucket* lebih sedikit dibandingkan dengan mengurutkan seluruh *array* secara langsung.

Kompleksitas algoritma *selection hybrid* dengan *bucket sort* yaitu, tahap pertama adalah mencari nilai maksimum dan minimum dalam *array*, yang membutuhkan waktu O(n). Kemudian, ukuran setiap *range* ditentukan dalam waktu konstan O(1). Selanjutnya, *array* kosong diinisialisasi untuk setiap *range* dengan kompleksitas waktu O(1). Tahap berikutnya adalah memasukkan setiap elemen ke dalam *range* yang sesuai. Ini dilakukan dalam waktu O(n), karena setiap elemen hanya perlu dimasukkan ke dalam satu *range* saja. Kemudian, setiap *range* diurutkan menggunakan *Selection sort hybrid* dengan kompleksitas waktu $O(n^2)$. Langkah terakhir menggabungkan setiap *range* menjadi satu *array* hasil akhir dengan kompleksitas waktu O(n). Pada kasus rata–rata terjadi ketika elemen-elemen *array* terdistribusi secara merata pada setiap *range* yang berukuran sedang dan *selection sort hybrid* digunakan untuk mengurutkan bilangan dalam *range* secara terbatas dan tidak secara keseluruhan bilangan dalam *range*, sehingga kompleksitas rata – rata algoritma gabungan *selection hybrid* dengan *bucket sort* adalah O(n log n). Pada kasus terburuk, Kompleksitas algoritma *selection hybrid* dengan *bucket sort* terjadi ketika semua elemen array berada pada satu range yang sama. Ini akan membuat algoritma ini bekerja seperti algoritma selection sort hybrid saja, dengan waktu kompleksitas $O(n^2)$. Pada kasus terbaik, setiap elemen array berada pada *range* sama yang berukuran kecil dan sudah terurut, sehingga tidak di perlukan *selection sort hybrid* untuk mengurutkan bilangan dalam array. Kompleksitas waktu pada kasus terbaik adalah O(n) dimana n adalah jumlah elemen pada array.

Pada penilitian ini, terdapat temuan baru dari penelitian sebelumnya, yaitu dalam penelitian sebelumnya, algoritma *selection sort* telah dimodifikasi menggunakan teknik *multithreading* untuk mengurutkan data secara keseluruhan (Hardika et al., 2020) tanpa dilakukan teknik pemisahan beberapa deret menjadi sub - deret, Namun pada penelitian ini digunakan modifikasi pada algoritma *selection sort hybrid* dan *bucket sort* dengan melakukan pembaguian menjadi beberapa sub-deret menggunakan algoritma *bucket sort* dan melakukan pengurutan elemen-elemen pada setiap sub-deret dengan *selection sort hybrid* yaitu menempatkan nilai minimum dengan nilai pada indeks awal *range*, dan menukarkan nilai maksimum pada indeks terakhir pada sub-deret. Pada penelitian sebelumnya (Gill et al., 2019), index *range* pada setiap *bucket* dimulai dari bilangan 0 hingga 10, pada penelitian ini index *range* di mulai dari nilai minimum yang di temukan pada deret bilangan hingga 10 sehingga dapat meminimalkan jumlah *bucket* atau *range* yang di pakai.

Selain itu, hasil penelitian ini juga menunjukkan bahwa penggunaan algoritma *bucket sort* dengan *selection sort hybrid* memberikan efisiensi pengurutan yang lebih tinggi daripada penggunaan algoritma merge sort (Nyoman et al., 2019). Pada penelitian sebelumnya, ketika banyak data diatas 100.000 bilangan *merge sort* memerkan waktu eksekusi diatas 0.30911 s sedangkan pada penelitian ini, algoritma gabungan *selection sort hybrid* dan *bucket sort* memiliki waktu eksekusi 0.109817 s. Hal ini dikarenakan selection sort hybrid hanya digunakan untuk mengurutkan elemen dalam array atau bucket yang telah di bagi menjadi beberapa sub-deret dengan *bucket sort*, sehingga penggunaannya dapat meningkatkan efisiensi

pengurutan pada setiap *range*. Penelitian lainnya dengan menggunakan algoritma *quick sort* (Poetra, 2022) untuk mengurutkan 1000 data, algoritma *quick sort* memerlukan waktu 0.00351 s sedangkan pada penelitian ini, algoritma gabungan *selection sort hybrid* dan *bucket sort* memerlukan waktu 0.001003 s. Sehingga algoritma gabungan *selection sort hybrid* dan *bucket sort* bisa mengurutkan data bilangan dengan lebih efisien dari algoritma *quick sort* dan *merge sort*.

SIMPULAN

Penggabungan antara algoritma *selection sort hybrid* dan *bucket sort* dapat memberikan efisiensi dan waktu eksekusi yang lebih baik dalam pengurutan data bilangan dibanding dengan algoritma lain seperti *Selection sort hybrid* biasa, algoritma *quick sort*, dan algoritma *merge sort* terutama jika jumlah data lebih dari 1000 bilangan. Selain itu, teknik pembagian sub-deret menggunakan algoritma bucket sort dengan index *range* dimulai dari nilai minimum pada deret bilangan dapat meminimalkan jumlah bucket atau *range* yang digunakan dalam pengurutan data. Hal ini dapat membantu meningkatkan efisiensi pengurutan pada setiap *range*. Dengan demikian, penggunaan algoritma gabungan selection sort hybrid dan bucket sort bisa menjadi alternatif yang lebih efisien dalam pengurutan data bilangan daripada algoritma *quick sort dan merge sort.*

REFERENSI

Adline, F., Tobing, T., & Tambunan, J. R. (2020). Analisis Perbandingan Efisiensi Algoritma Brute Force dan Divide and Conquer dalam Proses Pengurutan Angka. *Ultimatics: Jurnal Teknik Informatika*, 12(1), 52-58. https://doi.org/10.31937/ti.v12i1.1585

Andri, A. (2019). Penerapan Algoritma Pencarian Binary Search dan QuickSort pada Aplikasi Kamus Bahasa Palembang Berbasis Web. Jurnal Informatika: *Jurnal Pengembangan IT*, 4(1), 70–74. https://doi.org/10.30591/jpit.v4i1.1104

Anggreani, D., Wibawa, A. P., Purnawansyah, P., & Herman, H. (2020). Perbandingan Efisiensi Algoritma Sorting dalam Penggunaan Bandwidth. *ILKOM Jurnal Ilmiah*, 12(2), 96–103. https://doi.org/10.33096/ilkom.v12i2.538.96-103

Arifin, R. W., & Setiyadi, D. (2020). Algoritma Metode Pengurutan Bubble Sort dan Quick Sort Dalam Bahasa Pemrograman C++. *Information System For Educators And Professionals*, 4(2), 178–187.

Aqib, S. M., Nawaz, H., & Butt, S. M. (2021). Analysis of Merge Sort and Bubble Sort in Python, PHP, JavaScript, and C language. *International Journal of Advanced Trends in Computer Science and Engineering*, 10(2), 680–686. https://doi.org/10.30534/ijatcse/2021/311022021

Basir, R. R. (2020). Analisis Kompleksitas Ruang dan Waktu Terhadap Laju Pertumbuhan Algoritma Heap Sort, Insertion Sort dan Merge dengan Pemrograman Java. *STRING (Satuan Tulisan Riset dan Inovasi Teknologi)*, *5*(2), 109-118. http://dx.doi.org/10.30998/string.v5i2.6250

Chauhan, Y., & Duggal, A. (2020). Different sorting algorithms comparison based upon the time complexity. *International journal of research and analytical reviews (IJRAR)*.,7(3), 114-121.

Gill, S. K., Singh, V. P., Sharma, P., & Kumar, D. (2019). A comparative study of various sorting algorithms. *International Journal of Advanced Studies of Scientific Research*, 4(1). 367 -372 https://ssrn.com/abstract=3329410

Hardika, E., Atmaja, S., & Pinaryanto, K. (2020). Unjuk Kerja Selection Sort Hybrid 17. *Jurnal Buana Informatika*, 11(1), 17-25. https://doi.org/10.24002/jbi.v11i1.2699

Hasibuan, M. R. (2022). Implementasi Algoritma Quicksort Untuk Pembangkitan Kunci Algoitma RSA Pada Pengamanan Data Audio. *Journal of Informatics, Electrical and Electronics Engineering*, 2(1), 18-25. https://djournals.com/jieee

Hastomo, W., Karno, A. S. B., Kalbuana, N., Nisfiani, E., & Lussiana, E. T. P. (2021). Optimasi Deep Learning untuk Prediksi Saham di Masa Pandemi Covid-19. *JEPIN (Jurnal Edukasi dan Penelitian Informatika)*, *7*(2), 133-140. http://dx.doi.org/10.26418/jp.v7i2.47411

Ilmu, F., Dan, T., Uin, K., & Utara, S. (2021). Algoritma Pemrograman Dan Notasi Tertulis Retrieved April 14, 2023, from osf.io website: https://osf.io/fg2ca/download

Kumar, P., Gangal, A., Kumari, S., & Tiwari, S. (2020). Recombinant sort: N-dimensional cartesian spaced algorithm designed from synergetic combination of hashing, bucket, counting and radix sort. *Ingenierie Des Systemes d'Information*, 25(5), 655–668. https://doi.org/10.18280/ISI.250513

Nishom, M., & Fathoni, M. Y. (2018).Implementasi Pendekatan Rule-Of-Thumb untuk Optimasi Algoritma K-Means Clustering. *Jurnal Informatika: Jurnal Pengembangan IT*, 3(2), 237-241. http://dx.doi.org/10.30591/jpit.v3i2.909

Poetra, D. R. (2022). Performa Algoritma Bubble Sort Dan Quick Sort Pada Framework Flutter Dan Dart SDK (Studi Kasus Aplikasi E-Commerce). *JATISI (Jurnal Teknik Informatika Dan Sistem Informasi)*, *9*(2), 806-816.

Puspita Sari, Y., Ali, R., & Rajasa, A. (2022). Perbandingan Efisiensi dengan Algoritma Sorting dalam Penentuan Jarak (Studi Kasus: Pet Shop di Bandar Lampung). *TEKNIKA*, 16(1), 149-159.

Ramli, M. (2018). *Analisis Kunci Singkat Dinamis Algoritma One Time Pad untuk Keamanan Pes*an [Master's thesis, Universitas Sumatra Utara]. Repositori Universitas Sumatra Utara. http://repositori.usu.ac.id/handle/123456789/8158

Sari, N., Gunawan, W. A., Sari, P. K., Zikri, I., & Syahputra, A. (2022). Analisis Algoritma Bubble Sort Secara Ascending Dan Descending Serta Implementasinya Menggunakan Bahasa Pemrograman Java. *ADI Bisnis Digital Interdisiplin Jurnal*, 3(1), 16-23. https://doi.org/10.34306/abdi.v3i1.625

Setyantoro, D., & Hasibuan, R. A. (2020 Analisis dan perbandingan kompleksitas algoritma exchange sort dan insertion sort untuk pengurutan data menggunakan python *Tekinfo: Jurnal Bidang Teknik Industri dan Teknik Informatika*, 21(1), 48-56.

Toyib, R., Darnita, Y., Ragil, A., & Deva, S. (2021). Penerapan Algoritma Binary Search Pada Aplikasi E-Order. *Jurnal Media Infotama*, 17(1), 30 – 37 https://doi.org/10.37676/jmi.v17i1.1314

Tumanggor, H. Y., Maya, R., Lubis, F., Sianturi, M. P., Purba, R. G., & Manajemen, A. (2022). Metode algoritma bubble sort, algoritma merge sort dan algoritma quick sort dalam pengujian perbandingan proses penelitian kualitatif. *JUTISAL Jurnal Teknik Informatika Universa*l, 2(2), 47-58.

Utari, L. (2022). "algoritma selection sort" "implementasi algoritma selection sort untuk pengurutan nilai ipk mahasiswa universitas potensi utama." *JTIK (Jurnal Teknik Informatika Kaputama)*, 6(2), 390-398.

Wei, P., Huang, Z., Yang, X., & Jia, W. (2021). Fast adaptive phase unwrapping algorithm based on improved bucket sorting. *Optics and Lasers in Engineering*, 147(106745) 1 – 10. https://doi.org/10.1016/J.OPTLASENG.2021.106745