LAPORAN AKHIR
PROJECT DESAIN ANALISIS ALGORITMA
“PENCARIAN BUKU BERDASARKAN BINARY SEARCH DAN HASHING”

Dosen Pengampu :
Dr. Atik Wintarti, M.Kom.
Hasanuddin Al-Habib, M.Si.

Nama Anggota 8 Kelompok Sains Data 2022A:

1. Wawu Tri Ambodo. (22031554035)
2. Ahmad Hilmy Rakha A. (22031554052)
3. Reinesa Eveniashari P. (22031554021)

SAINS DATA 2022A

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM

UNIVERSITAS NEGERI SURABAYA

2023

I. Pendahuluan

Pencarian buku merupakan salah satu aktivitas yang umum dilakukan dalam pengelolaan perpustakaan atau sistem informasi. Dalam mengoptimalkan proses pencarian tersebut, beberapa metode pencarian dapat diterapkan, salah satunya adalah binary search dan hashing. Kedua metode ini memiliki keunggulan dan kelemahan masing-masing, namun keduanya bertujuan untuk mempercepat proses pencarian buku.

Binary search merupakan sebuah algoritma pencarian yang efisien pada data terurut. Dalam konteks pencarian buku, data buku biasanya diurutkan berdasarkan kategori, judul, atau penulis. Binary search memanfaatkan sifat data terurut ini untuk membagi data menjadi dua bagian pada setiap langkah pencarian, sehingga proses pencarian dapat dilakukan secara cepat dan efisien. Keunggulan binary search terletak pada kompleksitas waktu yang relatif rendah, terutama saat data buku sangat besar. Selain itu, penggunaan hashing juga menjadi alternatif yang populer dalam pencarian buku [1]. Hashing melibatkan penggunaan fungsi hash untuk mengonversi kunci pencarian (seperti nomor identifikasi buku) menjadi alamat atau indeks dalam struktur data yang disebut tabel hash. Metode ini memungkinkan pencarian buku dengan waktu yang konstan, karena setiap kunci pencarian diarahkan langsung ke lokasi yang sesuai dalam tabel hash. Namun, perlu diperhatikan bahwa konflik hashing dapat terjadi, yaitu saat dua atau lebih kunci memiliki nilai hash yang sama [2].

II. Metode

Data Collecting

Dataset yang kami gunakan merupakan dataset rekomendasi buku yang bersumber dari Kaggle yang terdiri dari 10018 baris dengan 14 kolom

title	series	author	book_link	genre	date_published	publisher	num_of_pag
The Martian	\n (The Martian #1)\n	Andy Weir	https://www.goodreads.com/book/show/18007564-t...	Science Fiction,Fiction,Audiobook,Adventure,Sp...	February 11th 2014	Crown\n\n ,	384
Under the Banner of Heaven: A Story of Violent...	NaN	Jon Krakauer	https://www.goodreads.com/book/show/10847.Unde...	Nonfiction,Religion,History,Crime,True Crime,M...	2004	Pan MacMillan\n\n ,	400
Cutting for Stone	NaN	Abraham Verghese	https://www.goodreads.com/book/show/3591262-cu...	Fiction,Historical,Historical Fiction,Cultural...	February 3rd 2009	Alfred A. Knopf\n\n ,	541
We Need to Talk About Kevin	NaN	Lionel Shriver	https://www.goodreads.com/book/show/80660.We_N...	Fiction,Contemporary,Thriller,Horror,Mystery,C...	July 3rd 2006	Harper Perennial\n\n ,	400
The Immortal Life of Henrietta Lacks	NaN	Rebecca Skloot	https://www.goodreads.com/book/show/6493208-th...	Nonfiction,Science,History,Biography,Health,Me...	February 2nd 2010	Crown Publishing Group\n\n ,	370

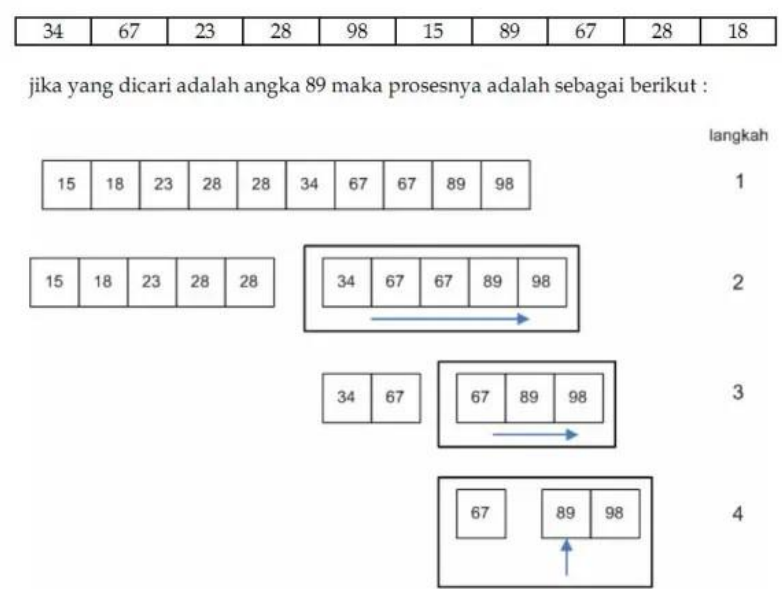
Gambar 1. dataset rekomendasi buku

Binary Search

Binary Search adalah algoritma dasar yang digunakan untuk menemukan nilai target secara efisien dalam array yang diurutkan. Algoritma ini bekerja dengan membagi berulang kali menjadi dua bagian array yang dapat berisi nilai target. Proses ini berlanjut hingga nilai ditemukan atau ruang pencarian kosong. Pencarian biner memiliki kompleksitas waktu sebesar O(log n), membuatnya jauh lebih cepat daripada pencarian linier untuk kumpulan data besar. Langkah-langkah dasar algoritma binary search adalah sebagai berikut:

1. Bandingkan nilai target dengan elemen tengah array.
2. Jika nilai target cocok dengan elemen tengah, posisinya dikembalikan.
3. Jika nilai target lebih kecil dari elemen tengah, maka pencarian dilanjutkan di bagian bawah array.
4. Jika nilai target lebih besar dari elemen tengah, maka pencarian dilanjutkan di paruh atas array.
5. Ulangi proses tersebut hingga nilai ditemukan atau ruang pencarian kosong.

binary search banyak digunakan dalam berbagai aplikasi, termasuk pencarian di database, menemukan elemen dalam array yang diurutkan, dan dalam banyak algoritma pembagian dan penaklukan. Ini adalah konsep kunci dalam ilmu komputer dan penting untuk memahami algoritma pencarian dan optimasi yang lebih kompleks.



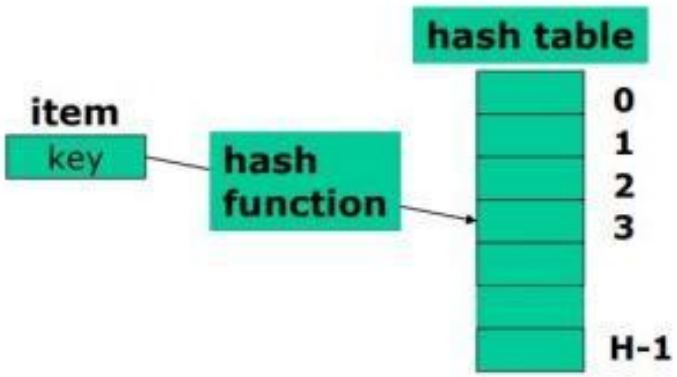
Gambar 2. ilustrasi operasi binary search

Hashing

Hashing adalah proses pengubahan data menjadi nilai yang lebih kecil, yang biasanya berupa nilai hash atau kode hash. Nilai hash ini kemudian digunakan untuk mengidentifikasi atau memetakan data asli. Proses hashing ini berguna dalam menyimpan dan mencari data

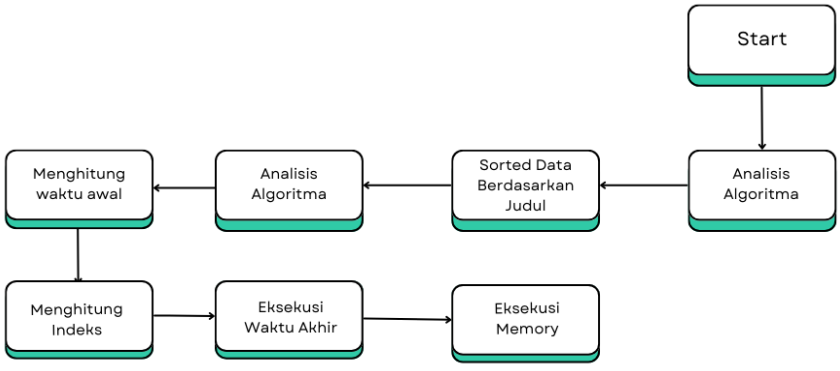
dengan cepat, karena nilai hash yang dihasilkan dapat digunakan sebagai alamat atau petunjuk untuk menemukan data yang sesuai.

Metode hashing umumnya digunakan dalam struktur data seperti tabel hash, yang memungkinkan pencarian data dengan kompleksitas waktu yang lebih rendah daripada metode pencarian linier. Dalam proses hashing, terdapat kemungkinan terjadinya tabrakan hash, yaitu ketika dua data yang berbeda menghasilkan nilai hash yang sama. Untuk mengatasi hal ini, teknik penanganan tabrakan seperti chaining atau open addressing dapat diterapkan. Dalam implementasinya, hashing banyak digunakan dalam berbagai aplikasi, seperti basis data, keamanan data, dan pengindeksan data. Kecepatan pencarian data yang dihasilkan oleh metode hashing membuatnya menjadi pilihan yang populer dalam pengembangan sistem informasi [3].



Gambar 3. ilustrasi operasi algoritma Hashing

Diagram Alir



Gambar 4. Diagram alir project

1. Start

Aliran dimulai dengan tanda "Start," menandakan awal dari algoritma

2. Analisis Algoritma

Proses ini mencakup analisis algoritma yang akan digunakan. Ini melibatkan pemahaman terhadap langkah-langkah yang diperlukan untuk mencapai tujuan, dalam hal ini, pengurutan data berdasarkan judul

3. Sorted Data Berdasarkan Judul

Langkah ini melibatkan pengurutan data berdasarkan judulnya. Algoritma pengurutan yang digunakan, seperti pengurutan cepat atau pengurutan gabungan, akan tergantung pada kebutuhan dan kompleksitas waktu yang diinginkan.

4. Menghitung Waktu Awal

Pada langkah ini, waktu awal atau timestamp dihitung. Ini bertujuan untuk memantau kapan proses pengurutan dimulai

5. Menghitung Indeks

Langkah ini melibatkan perhitungan indeks yang diperlukan untuk mengakses elemen-elemen data selama proses pengurutan. Hal ini mungkin mencakup penghitungan indeks awal dan indeks akhir, tergantung pada algoritma pengurutan yang digunakan

6. Eksekusi Waktu Akhir

Proses pengurutan data sekarang dieksekusi, dan waktu akhir atau timestamp untuk menandai akhir proses pengurutan dihitung. Ini membantu dalam mengukur lama waktu yang diperlukan untuk menyelesaikan pengurutan.

7. Eksekusi Memori

Langkah terakhir mencakup analisis kompleksitas memori dari algoritma pengurutan. Ini melibatkan penilaian terhadap berapa banyak memori yang diperlukan selama eksekusi, termasuk alokasi memori untuk struktur data tambahan, variabel, dan proses lainnya.

III. Hasil dan Pembahasan

Implementasi Binary Search:
Melakukan eksperimen dengan menggunakan metode Binary Search pada dataset buku yang telah terurut (misalnya, berdasarkan judul buku atau penulis).
Mengukur waktu yang diperlukan untuk menemukan buku tertentu menggunakan Binary Search.

Implementasi Hashing:
Menerapkan metode Hashing dengan membuat struktur data tabel hash.
Memasukkan data buku ke dalam tabel hash menggunakan fungsi hash yang sesuai, seperti berdasarkan nomor identifikasi buku.
Mengukur waktu pencarian buku menggunakan metode hashing dari tabel hash yang telah dibuat.

Perbandingan Efisiensi dan Keefektifan:
Membandingkan waktu yang diperlukan untuk menemukan buku dengan menggunakan kedua metode.
Menganalisis efisiensi keduanya pada berbagai skenario, termasuk data buku yang berukuran berbeda dan pola pencarian yang berbeda.

Penanganan Konflik Hashing:
Memeriksa dan menganalisis penanganan konflik hashing dalam konteks dataset buku yang digunakan.
Menyediakan rekomendasi atau solusi untuk mengatasi konflik hashing yang mungkin terjadi.

Pengukuran Kinerja:
Menghitung kompleksitas waktu dari kedua metode pencarian berdasarkan hasil eksperimen dan analisis yang telah dilakukan.
Membuat grafik atau tabel perbandingan yang menunjukkan keunggulan dan kelemahan masing-masing metode dalam situasi yang berbeda.

A. Analisis Big-O

Binary Search

Langkah-langkah analisis Big O dari kode Binary Search

1. Input
- `n`: Jumlah elemen dalam array yang diurutkan.

2. Langkah Pertama
- Inisialisasi variabel `low` dan `high` sebagai indeks pertama dan terakhir dalam array.

3. Iterasi Melalui Binary Search
- Selama `low` kurang dari atau sama dengan `high`:
- Hitung indeks tengah: `mid = (low + high) // 2`.
- Bandingkan elemen tengah dengan elemen yang dicari.
- Jika elemen tengah sama dengan elemen yang dicari, kembalikan indeks tengah.
- Jika elemen tengah kurang dari elemen yang dicari, atur `low = mid + 1`.
- Jika elemen tengah lebih besar dari elemen yang dicari, atur `high = mid - 1`.

4. Analisis Kompleksitas Waktu (Big O)
- Pada setiap iterasi, ukuran ruang pencarian (range) diabaikan setengahnya, karena hanya satu sisi yang tetap dipertahankan (meninggalkan setengah lainnya).
- Kompleksitas waktu Binary Search adalah O(log n), dimana `n` adalah jumlah elemen dalam array.

- Ini menunjukkan bahwa algoritma Binary Search memiliki pertumbuhan waktu yang sangat lambat seiring dengan peningkatan ukuran array, karena jumlah langkah yang diperlukan hanya logaritmik terhadap ukuran array.

Hashing

Langkah-langkah Analisis Big O dari kode Hashing:

1. Inisialisasi
- Inisialisasi tabel hash dan fungsi hash.
- `n`: Jumlah elemen yang akan dimasukkan ke dalam tabel hash.
2. Input
- `key`: Kunci pencarian atau identifikasi unik dari setiap elemen.
- `value`: Nilai atau data yang akan disimpan.
3. Fungsi Hash
- Hitung nilai hash dari kunci menggunakan fungsi hash.
- Set `index = hash value % size`, di mana `size` adalah ukuran tabel hash.
4. Penanganan Konflik
- Jika terjadi konflik (dua atau lebih kunci hash ke indeks yang sama)
- Penanganan konflik dapat dilakukan dengan chaining (menggunakan linked list untuk menyimpan elemen yang berkonflik di indeks yang sama) atau metode penanganan konflik lainnya.
5. Analisis Kompleksitas Waktu (Big O)
- Fungsi hash: O(1) jika fungsi hash memiliki kompleksitas waktu tetap. Namun, fungsi hash yang kurang efisien dapat meningkatkan kompleksitas waktu.
- Penanganan konflik: Tergantung pada metode yang digunakan. Chaining biasanya memiliki kompleksitas waktu O(1) untuk menyisipkan dan mengakses elemen di linked list.

B. Analisis Waktu

Binary search

Analisis kompleksitas waktu dari algoritma binary search menunjukkan bahwa, baik diimplementasikan secara iteratif maupun rekursif, kompleksitasnya adalah O(log N), di mana N adalah ukuran array atau struktur data yang telah diurutkan. Algoritma ini efisien karena setiap langkah membandingkan elemen tengah dengan elemen yang dicari, mengurangi setengah dari area pencarian pada setiap iterasi. Dengan demikian, binary search memiliki kompleksitas waktu non-linear yang memberikan efisiensi pencarian yang tinggi, terutama pada data yang besar, dan menjadikannya salah satu algoritma pencarian yang paling efisien untuk data yang telah diurutkan.

Hashing

Waktu yang diperlukan untuk menghitung nilai hash dari suatu elemen dengan fungsi hash diasumsikan O(1) dalam kasus rata-rata, tetapi perlu diperhatikan bahwa implementasi yang buruk atau konflik hash dapat meningkatkan kompleksitas ini. Penanganan kolisi, baik melalui chaining, probing, atau metode lainnya, memengaruhi kompleksitas waktu secara keseluruhan. Akses ke tabel hash, baik untuk memasukkan atau mengambil elemen, sering dianggap sebagai O(1) dalam kasus rata-rata, namun, faktor seperti penyimpanan di cache atau struktur data tambahan dapat mempengaruhi

kompleksitas ini. Selain itu, kompleksitas waktu dapat dipengaruhi oleh ukuran tabel hash, dimana semakin besar ukuran tabel dapat mengurangi kemungkinan kolisi, namun, dapat mempengaruhi efisiensi dalam penggunaan memori. Secara umum, analisis yang lebih mendetail tergantung pada implementasi khusus dan metode penanganan kolisi yang digunakan.

C. Analisis Memory

Binary Search

algoritma binary search melibatkan penggunaan dan penyimpanan data di dalam memori. Binary search biasanya diterapkan pada array atau struktur data yang diindeks secara langsung. Variabel seperti indeks pertama, terakhir, dan tengah memerlukan ruang memori konstan (O(1)). Jika diimplementasikan secara rekursif, binary search memiliki kompleksitas memori O(log N) karena penggunaan stack untuk rekursi dengan kedalaman logaritmik terhadap ukuran array. Dengan demikian, kompleksitas memori keseluruhan tergantung pada implementasi dan bisa menjadi O(1) untuk versi non-rekursif atau O(log N) untuk versi rekursif.

Hashing

algoritma hashing melibatkan alokasi memori untuk tabel hash dengan ukuran konstan (O(m)), di mana m adalah ukuran tabel hash. Memori tambahan dibutuhkan untuk menyimpan elemen-elemen yang di-hash, tergantung pada jenis data dan struktur penyimpanannya. Penanganan kolisi, jika diterapkan, memerlukan memori tambahan untuk struktur data yang menangani elemen-elemen yang bertabrakan. Fungsi hash memerlukan ruang memori konstan (O(1)). Analisis memori ini bersifat umum dan dapat dipengaruhi oleh implementasi khusus, alokasi memori dinamis, serta metode penanganan kolisi yang digunakan.

Komparasi atau Perbandingan Kedua Algoritma

	Binary Search	Hashing
Pencarian ditemukan di Indeks	1734	1735
Waktu eksekusi	0.0080263 detik	0.00501751 detik
Memory	32.00 KB	1848.00 KB
Worst Case	O(log n)	O(n)

Tabel 1. Komparasi perbandingan dua Algoritma

```
Element Cutting for Stone found at index 1734
Time taken for Binary Search: 0.008026361465454102 seconds
Memory used for Binary Search: 32.00 KB
```

Gambar 5. Hasil Eksekusi Algoritma Binary Search

Gambar 6. Hasil Eksekusi Algoritma Hashing

```
Element Cutting for Ston not found
Time taken for Hashing-based Search: 0.011009693145751953 seconds
Memory used for Hashing-based Search: -760.00 KB
Number of iterations: 0
```

Gambar 7. Hasil Eksekusi Algoritma Hashing jika terjadi worst case

```
Element Cutting for Ston not found
Time taken for Hashing-based Search: 0.004000425338745117 seconds
Memory used for Hashing-based Search: -244.00 KB
Number of iterations: 0
```

Gambar 8. Hasil Eksekusi Algoritma Binary Search jika terjadi worst case

Analisis Hasil

Berdasarkan tabel komparasi, terlihat bahwa terjadi perbedaan di pencarian Judul buku ditemukan di indeks yang berbeda, mencari buku dengan judul Cutting for Stone di dua algoritma yakni Binary ditemukan di indeks ke 1734 sedangkan di algoritma Hashing ditemukan di indeks ke 1735. Waktu pencarian relatif rendah dapat dicapai oleh algoritma binary search ketika dapat menemukan elemen dalam jumlah iterasi yang sedikit, terutama pada dataset besar. Algoritma ini menjadi lumayan efektif pada dataset yang sudah diurutkan, karena dengan adanya pengurutan, binary search dapat meminimalkan jumlah iterasi yang diperlukan untuk menemukan elemen target. Selain itu, kelebihan algoritma ini juga terletak pada penggunaan memori yang relatif rendah, sehingga membuatnya menjadi pilihan yang efisien untuk pencarian dalam dataset yang besar dan terurut. Pada algoritma Hashing Waktu pencarian konstan, di mana pencarian dilakukan dalam waktu tetap tanpa ketergantungan pada ukuran dataset, menandakan kecepatan tinggi dalam pencarian. Algoritma hashing, dengan pengindeksan langsung ke alamat, menjadi ideal untuk pencarian dengan skala besar. Selain itu, algoritma ini menawarkan penyimpanan yang efisien dengan struktur tabel hash yang sederhana. Meskipun waktu pencarian konstan dan kecepatan tinggi adalah keunggulan utama, algoritma hashing juga efektif untuk dataset yang cukup besar dan memerlukan manajemen penyimpanan yang efisien.

Waktu pencarian algoritma binary search memiliki kompleksitas O(log n), yang terjadi ketika elemen yang dicari berada di ujung dataset atau tidak ada dalam dataset. Dalam kasus ini, setiap iterasi membagi dataset menjadi setengah, sehingga jumlah iterasi berkurang secara eksponensial, memberikan efisiensi tinggi dalam menemukan elemen. Sebaliknya, waktu pencarian algoritma hashing memiliki kompleksitas O(n) dalam kasus terburuk. Ini terjadi ketika terjadi konflik hashing yang ekstensif, menyebabkan semua kunci hash berada di dalam satu slot tabel hash. Pada situasi ini, pencarian menjadi linier, memerlukan iterasi melalui seluruh slot tabel, mengurangi efisiensi secara signifikan. Oleh karena itu, sementara binary search sangat efisien untuk pencarian dalam dataset yang terurut, algoritma hashing memiliki

kinerja yang lebih baik ketika konflik hashing dapat diminimalkan untuk mencapai pencarian yang optimal.

IV. Kesimpulan

Dalam pengelolaan perpustakaan atau sistem informasi, pencarian buku menjadi kegiatan umum. Dua metode pencarian yang umum digunakan adalah binary search dan hashing, keduanya dengan tujuan mempercepat proses pencarian buku. Binary search, efisien pada data terurut, membagi dataset secara eksponensial pada setiap langkahnya dengan kompleksitas waktu O(log n). Keunggulannya terletak pada efisiensi pencarian yang tinggi, terutama pada data yang besar. Sementara itu, hashing menggunakan fungsi hash untuk mengonversi kunci pencarian menjadi alamat atau indeks dalam tabel hash. Meskipun menawarkan pencarian konstan, kompleksitas waktu tergantung pada penanganan konflik hashing. Analisis eksperimen menunjukkan bahwa binary search efektif untuk dataset terurut, sementara hashing cocok untuk skala besar dengan penanganan konflik yang optimal.

Implementasi kedua algoritma pada dataset buku menghasilkan perbandingan waktu pencarian yang menarik. Binary search dan hashing memberikan hasil yang berbeda dalam menemukan elemen di indeks tertentu, dengan keunggulan masing-masing. Binary search memiliki kompleksitas waktu yang lebih baik dalam kasus terurut, sementara hashing menawarkan waktu pencarian konstan, terutama pada skala besar. Perbedaan juga terlihat dalam penggunaan memori, di mana binary search cenderung lebih efisien daripada hashing dalam hal ini. Pemilihan antara binary search dan hashing tergantung pada karakteristik dataset dan kebutuhan pencarian. Binary search efektif untuk data terurut dengan kompleksitas waktu yang relatif rendah, sedangkan hashing menawarkan kecepatan konstan dalam pencarian. Oleh karena itu, pemahaman mendalam tentang kelebihan dan kelemahan masing-masing algoritma menjadi kunci dalam memilih pendekatan yang sesuai dengan kebutuhan sistem informasi atau perpustakaan.

V. Pembagian Tugas

1. Wawu Tri Ambodo 22031554035
- Merancang Code Binary Search
- Analisis Algoritma
- Membuat Laporan
- Membuat PPT

2. Ahmad Hilmy Rakha A. 22031554052
- Merancang Code Hashing
- Analisis Algoritma
- Membuat Laporan
- Membuat PPT

3. Reinesa Eveniashari P. 22031554021
- Analisis Algoritma
- Membuat Laporan
- Membuat PPT

VI. Referensi

[1] D. Amato, G. Lo Bosco, and R. Giancarlo, “Standard versus uniform binary search and their variants in learned static indexing: The case of the searching on sorted data benchmarking software platform,” *Softw Pract Exp*, vol. 53, no. 2, pp. 318–346, Feb. 2023, doi: 10.1002/spe.3150.

[2] X. Chen, Y. Li, and C. Chen, “An Online Hashing Algorithm for Image Retrieval Based on Optical-Sensor Network,” *Sensors*, vol. 23, no. 5, Mar. 2023, doi: 10.3390/s23052576.

[3] L. Du, Z. He, Y. Wang, X. Wang, and A. T. S. Ho, “An image hashing algorithm for authentication with multi-attack reference generation and adaptive thresholding,” *Algorithms*, vol. 13, no. 9, Sep. 2020, doi: 10.3390/A13090227.

Lampiran Listing Code

1. Binary Search

```
def binary_search(arr, target):
    low, high = 0, len(arr) - 1
    iterations = 0

    while low <= high:
        mid = (low + high) // 2
        mid_val = arr[mid]
        iterations += 1

        if mid_val.lower() == target.lower():
            return mid, iterations
        elif mid_val.lower() < target.lower():
            low = mid + 1
        else:
            high = mid - 1

    return -1, iterations

# Mengurutkan judul sebelum pencarian
judul = sorted(df["title"])

# Mengukur waktu sebelum eksekusi
start_time = time.time()

# Mengukur penggunaan memori sebelum eksekusi
start_memory = psutil.virtual_memory().used

# Penggunaan
target = "Cutting for Stone"
result, iterations = binary_search(judul, target)

# Mengukur waktu setelah eksekusi
end_time = time.time()

# Mengukur penggunaan memori setelah eksekusi
end_memory = psutil.virtual_memory().used

if result != -1:
    print(f"Element {target} found at index {result}")
else:
    print(f"Element {target} not found")

# Cetak waktu yang dibutuhkan untuk pencarian
print(f"Time taken for Binary Search: {end_time - start_time} seconds")
```

```
# Cetak penggunaan memori dalam kilobytes
print(f"Memory used for Binary Search: {((end_memory - start_memory) /
1024):.2f} KB")
print(f"Number of iterations: {iterations}")
```

2. Hashing

```
def create_hash_table(genres):
    hash_table = {}
    for index, genre in enumerate(genres):
        hash_key = hash(genre)
        hash_table[hash_key] = index
    return hash_table

def hash_search(hash_table, target):
    hash_key = hash(target)
    iterations = 0
    index = hash_table.get(hash_key, -1)

    while index == -1 and hash_key in hash_table:
        hash_key += 1  # Linear probing for handling collisions
        index = hash_table.get(hash_key, -1)
        iterations += 1

    return index, iterations

# Mengurutkan judul sebelum pencarian
judul = sorted(df["title"])

# Creating a hash table
hash_table = create_hash_table(judul)

# Mengukur waktu sebelum eksekusi
start_time = time.time()

# Mengukur penggunaan memori sebelum eksekusi
start_memory = psutil.virtual_memory().used

# User input for the target genre
target = "Cutting for Stone"
result, iterations = hash_search(hash_table, target)

# Measuring time after execution
end_time = time.time()

# Mengukur penggunaan memori setelah eksekusi
end_memory = psutil.virtual_memory().used

if result != -1:
```

```
    print(f"Element {target} found at index {result}")
else:
    print(f"Element {target} not found")

# Print the time taken for hashing-based search
print(f"Time taken for Hashing-based Search: {end_time - start_time} seconds")

# Cetak penggunaan memori dalam kilobytes
print(f"Memory used for Hashing-based Search: {((end_memory - start_memory) /
1024):.2f} KB")

# Print the number of iterations
print(f"Number of iterations: {iterations}")
```