Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

INTERPOLATION SEARCHING ALGORITHM VS ALGORITMA PENCARIAN TRADISIONAL: ANALISIS EFISIENSI MEMORI DAN WAKTU KOMPUTASI

Nazhifa Naura Yasmin[1], Lailatul Sofia[2], Arum Rahma Putri Sabrina[3], Adhystha Az-Zahra Putri[4], Imam Prayogo Pujiono[5]
[1]nazhifa.naura.yasmin24022@mhs.uingusdur.ac.id, [2]lailatul.sofia24027@mhs.uingusdur.ac.id, [3]arum.rahma.putri.sabrina24028@mhs.uingusdur.ac.id, [4]adhysta.az-zahra.putri24024@mhs.uingusdur.ac.id, [5]imam.prayogopujiono@uingusdur.ac.id
[1,2,3,4,5]Informatika, UIN K.H. Abdurrahman Wahid Pekalongan

Abstrak

Penelitian ini bertujuan untuk membandingkan efisiensi memori dan waktu komputasi tiga algoritma pencarian: *Sequential Search*, *Binary Search*, dan *Interpolation Search* dalam bahasa C++. Pengujian dilakukan pada dataset terurut tanpa duplikasi sebanyak 100, 1.000, dan 10.000 data. Hasil menunjukkan *Sequential Search* paling hemat memori dengan rata-rata 12 byte pada semua dataset. *Binary Search* mencatat konsumsi memori tertinggi, terutama pada data besar (112 byte). Dari sisi waktu komputasi, *Interpolation Search* unggul pada dataset sedang (0,011 detik), sedangkan *Sequential Search* lebih cepat pada dataset kecil dan besar (0,019 dan 0,018 detik). *Binary Search* menunjukkan waktu komputasi paling tinggi (hingga 0,040 detik). Hasil ini menunjukkan bahwa pemilihan algoritma perlu mempertimbangkan ukuran dan pola distribusi data untuk mencapai efisiensi optimal.

Kata kunci: Algoritma Pencarian, Efisiensi Memori, Waktu Komputasi, C++, Interpolation Search

Abstract

This study aims to compare the memory efficiency and computation time of three search algorithms: Sequential Search, Binary Search, and Interpolation Search in C++. Tests were conducted on sorted datasets without duplication of 100, 1,000, and 10,000 data. Results show Sequential Search is the most memory-efficient with an average of 12 bytes on all datasets. Binary Search recorded the highest memory consumption, especially on large data (112 bytes). In terms of computation time, Interpolation Search excelled on medium datasets (0.011 seconds), while Sequential Search was faster on small and large datasets (0.019 and 0.018 seconds). Binary Search showed the highest computation time (up to 0.040 seconds). These results suggest that algorithm selection needs to consider the size and distribution pattern of the data to achieve optimal efficiency.

Keywords: *Search algorithm, Memory Efficiency, Computation Time, C++, Interpolation Search*

1. Pendahuluan

Search merupakan aktivitas yang sering dilakukan dalam kehidupan sehari-hari. Terkadang, tujuan *search* hanya untuk memastikan keberadaan suatu data dalam sekumpulan data [1]. Namun, di waktu lain, *search* dilakukan untuk mengetahui posisi atau letak data yang dicari [2]. Algoritma *search* merupakan pondasi penting dalam berbagai sistem, mulai dari *search engine*, manajemen basis data, hingga aplikasi sehari-hari seperti pencarian dokumen atau file [3]. Dengan adanya algoritma ini, proses *search* menjadi lebih cepat dan terstruktur, sehingga memudahkan pengguna dalam memperoleh informasi yang dibutuhkan secara lebih efisien [4].

Dalam ilmu komputer, algoritma *search* memiliki peran penting dalam meningkatkan efisiensi pencarian data, terutama pada data yang besar dan kompleks [5]. Terdapat berbagai algoritma *search* yang telah dikembangkan, di antaranya adalah algoritma *search* tradisional (*Sequential Search* dan *Binary Search)* dan Algoritma *Modern* (*Interpolation Search*) [6]. Algoritma *Sequential Search* merupakan metode *Search* sederhana yang bekerja dengan memeriksa elemen satu per satu hingga elemen yang dicari ditemukan atau hingga akhir data tercapai [7]. Algoritma *Sequential Search* merupakan tindakan yang dilakukan satu persatu sesuai dengan urutan penulisan, jika susunannya diubah maka hasil yang diperoleh

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

kemungkinan besar akan berbeda [8]. Metode *Binary Search* menggunakan pendekatan pembagian dua, di mana data dibagi menjadi dua bagian, lalu dibandingkan untuk menentukan lokasi elemen yang dicari [9]. Sedangkan *Interpolation Search* memperkirakan posisi elemen yang dicari berdasarkan pola distribusi data, sehingga memungkinkan *search* yang lebih cepat dalam kondisi tertentu [2].

Algoritma *Sequential Search* merupakan metode *search* yang dilakukan secara serentak dari kedua ujung daftar menuju tengah untuk menemukan elemen yang dicari [10]. Pendekatan ini membuat skenario kasus terburuknya lebih efisien dibandingkan algoritma *search* lainnya. Dalam kondisi tertentu, seperti saat elemen yang dicari berada di posisi paling akhir dalam *array*, *Sequential Search* dapat melampaui kecepatan *Binary Search* dengan kompleksitas waktu mencapai ***O(1)*** [11].

Analisis algoritma *search* merupakan proses untuk mengevaluasi seberapa efisien suatu algoritma dalam hal kecepatan eksekusi (kompleksitas waktu) dan penggunaan memori (kompleksitas ruang). Memahami analisis ini menjadi sangat penting, sebab memilih algoritma yang kurang tepat dapat menyebabkan penurunan performa, terlebih saat menangani data dalam jumlah besar. Penelitian sebelumnya telah banyak mengkaji algoritma *Sequential Search*, *Binary Search*, dan *Interpolation Search* dalam berbagai skenario *search* data. Namun, kebanyakan studi lebih menyoroti implementasi tanpa membahas secara mendalam aspek efisiensi memori dan waktu komputasi pada lingkungan pemrograman C++ [1].

*Sequential Searc*h merupakan algoritma yang paling sederhana yang dapat memeriksa setiap elemen dengan data yang berurutan hingga menemukan elemen yang dicari [12]. Pada kasus terbaik, elemen ditemukan diindeks pertama dengan kompleksitas waktu ***O(1)***, tetapi jika pada kasus yang mempunyai rata-rata terburuk, *search* bisa mencapai hingga akhir daftar dengan kompleksitas waktu ***O(n)***, dimana ***n*** adalah jumlah elemen [13]. Sedangkan, *Binary Search* bekerja dengan membagi daftar yang sudah diurutkan menjadi dua bagian dan memilih bagian yang relevan untuk dilanjutkan [14]. Algoritma ini sangat efisien untuk digunakan Dataset besar yang terurut, dengan kompleksitas waktu pada kasus terbaik ***O(log n)***, dan pada kasus terburuk tetap berada pada di ***O(log n)*** [15]. Berbeda dengan *Sequential Search* dan *Binary Search*, *Interpolation Search* memilih menggunakan pendekatan estimasi posisi berdasarkan nilai kunci yang dicari. Jika distribusi data seragam, algoritma memiliki kompleksitas waktu rata-rata ***O(log log n)***, lebih cepat dari *Binary Search* [16]. Tetapi, dalam kondisi data yang tidak merata, performa *Interpolation Search* bisa turun drastis hingga ***O(n)***, sama seperti kasus terburuk *Sequential Search* [17].

Penelitian sebelumnya telah mengkaji implementasi algoritma pencarian, khususnya *Sequential Search* dan *Binary Search*, dalam berbagai konteks aplikasi. Pada tahun 2021, Wenni Lita Yuniar dan Fatkhul Amin melakukan penelitian mengenai sistem pencarian naskah dinas di Polres Kendal dengan menggunakan algoritma *Sequential Search*. Penelitian ini bertujuan untuk mempercepat proses pencarian yang sebelumnya dilakukan secara konvensional dan memakan waktu cukup lama [18]. Selanjutnya, pada tahun 2022, Dian Markuci dan Cahyo Prianto membandingkan performa algoritma *Sequential Search* dan *Binary Search* dalam aplikasi surat perjalanan dinas. Hasil penelitian menunjukkan bahwa algoritma *Binary Search* memiliki performa pencarian yang lebih cepat dan stabil dibandingkan *Sequential Search*. Namun, dalam kondisi tertentu, terutama ketika data berada di posisi awal, *Sequential Search* dapat memberikan hasil yang lebih cepat. Penelitian ini juga menunjukkan bahwa jumlah data mempengaruhi performa kecepatan pencarian, di mana *Sequential Search* cenderung mengalami penurunan efisiensi seiring bertambahnya jumlah data, sedangkan *Binary Search* tetap menunjukkan kecepatan pencarian yang konsisten karena memanfaatkan data yang sudah terurut dan pembagian ruang pencarian secara sistematis [19].

Pada penelitian ini dilakukan perbandingan antara *Interpolation Search* dengan dua algoritma *search* tradisional yaitu *Binary Search* dan *Sequential Search*, perbandingan ini akan membandingkan efisiensi memori dan waktu komputasi dari ketiga algoritma tersebut dengan menggunakan bahasa pemrograman C++ [20]. Data input menggunakan tipe data numerik acak tetapi dengan nilai terurut dan tidak ada duplikasi dalam tiga ukuran dataset yang berbeda: 100, 1.000, dan 10.000. Efisiensi memori dihitung berdasarkan selisih antara memori yang digunakan setelah eksekusi dengan memori yang digunakan sebelum eksekusi, sedangkan waktu komputasi dihitung dari durasi yang dibutuhkan komputer untuk menyelesaikan proses. Data input yang bervariasi digunakan untuk menguji kinerja ketiga algoritma dalam proses *search* dengan *random target* sehingga *search* untuk target data numerik dicari oleh sistem, dengan data kecil (100), sedang (1.000), dan besar (10.000). Selanjutnya kinerja *Interpolation Search* akan dianalisis dan dibandingkan dengan kedua algoritma *search* untuk menentukan apakah *Interpolation*

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

Search lebih efisien dalam hal penggunaan memori dan waktu komputasi pada *search* data berjumlah 100, 1.000, dan 10.000 [21].

2. Metode

Tahapan penelitian ini dimulai dengan pembuatan dataset dalam tiga skala berbeda, yaitu data kecil sebanyak 100 data, data sedang sebanyak 1.000 data, dan data besar sebanyak 10.000 data. Dataset yang digunakan terdiri dari bilangan acak dengan nilai unik dan telah disusun secara *ascending*. Rentang nilai acak yang digunakan disesuaikan dengan ukuran data, yakni 0–200 untuk dataset kecil, 0–2.000 untuk dataset sedang, dan 0–20.000 untuk dataset besar. Karakteristik dataset berdasarkan jumlah elemen dan rentang nilai acak dapat dilihat pada Tabel 1.

Tabel 1. Deskripsi Dataset Pengujian

Skala Dataset	Jumlah data	Rentang nilai	Keterangan
Kecil	100	0-200	Data unik, ascending
Sedang	1000	0-2000	Data unik, ascending
Besar	10000	0-20000	Data unik, ascending

Proses pembuatan data dilakukan menggunakan bahasa pemrograman C++, dengan memanfaatkan struktur *unordered_set* untuk menjamin bahwa tidak ada angka yang duplikat. Berikut adalah potongan fungsi program yang digunakan:

```
// Fungsi untuk membuat dataset angka acak unik terurut
// jumlahData disesuaikan: 100, 1000, atau 10000
void generateUniqueRandomThenSort(int jumlahData) {
    std::unordered_set<int> dataSet; // Membuat set untuk menyimpan angka unik
    srand(time(0) + jumlahData); // Seed random berdasarkan waktu dan jumlah data

    while (dataSet.size() < jumlahData) { // Ulangi hingga set berisi jumlahData elemen
        int angka = rand() % 200; // Menghasilkan angka acak 0-200 (batas atas disesuaikan dataset)
        dataSet.insert(angka); // Menambahkan angka ke set, otomatis mencegah duplikat
    }

    std::vector<int> data(dataSet.begin(), dataSet.end()); // Memindahkan set ke vector
    std::sort(data.begin(), data.end()); // Mengurutkan vector secara ascending
}
```

Fungsi ini bekerja dengan menghasilkan bilangan acak yang tidak berulang, menyesuaikan batas maksimum dengan skala dataset, dan mengurutkannya untuk memastikan dataset siap digunakan pada algoritma *search*. Dataset hasil dari program tersebut digunakan sebagai masukan untuk pengujian dua algoritma *search*, yaitu *Binary Search* dan *Interpolation Search*. Masing-masing algoritma diuji pada tiga skala dataset: 100, 1.000, dan 10.000 data.

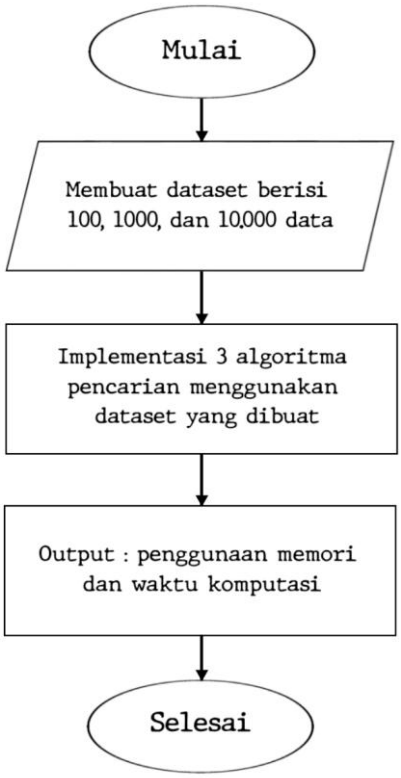
Implementasi pengujian algoritma *search* ini dilakukan menggunakan bahasa pemrograman C++, dan bertujuan untuk mengukur dan membandingkan performa algoritma berdasarkan beberapa indikator, yaitu:

1. Waktu eksekusi terhadap berbagai ukuran data.
2. Penggunaan memori selama proses *search*.
3. Evaluasi efektivitas dan efisiensi algoritma dalam beragam kondisi data.

Data hasil pengujian akan dianalisis untuk mengidentifikasi kelebihan dan kekurangan dari masing-masing algoritma. Selanjutnya, penelitian ini akan merumuskan rekomendasi mengenai teknik optimasi yang dapat diterapkan guna meningkatkan kinerja algoritma *search*, baik dalam mengurangi kompleksitas waktu maupun mengoptimalkan penggunaan ruang.

Seluruh hasil pengujian akan disajikan dalam bentuk tabel dan visualisasi grafik guna mempermudah proses perbandingan performa kedua algoritma. Gambar 1 memberikan ilustrasi dari keseluruhan tahapan penelitian ini.

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857



Gambar 1. Diagram Alur Tahapan Penelitian

3. Hasil dan Pembahasan

Penelitian ini dilakukan memakai bahasa pemrograman C++ dengan menggunakan aplikasi *CodeBlocks* untuk mengembangkan serta menjalankan kode program. Pengujian dilakukan pada laptop dengan spesifikasi sebagai berikut:

1. Prosesor Intel Core i5-1235U 1.30GHz
2. Memori 16 GB RAM
3. Sistem operasi Windows 11 64-Bit
4. SSD berkapasitas 512GB

Pada pengujian ini melibatkan algoritma *search* tradisional yang terdiri dari *Binary Search* dan *Sequential Search*, *search* tradisional tersebut nantinya akan diuji dengan *Interpolation Search*. Ketiga algoritma diuji menggunakan dataset yang sama, terdiri dari 100 data dataset kecil) dengan rentang nilai 0-200, 1.000 data (dataset sedang) dengan rentang nilai 0-2.000, dan 10.000 data (dataset besar) dengan rentang nilai 0-20.000 setiap data bertipe *ascending* dan *unordered-set* (tidak ada angka yang duplikat). Setiap algoritma diuji dengan tiga kali pengujian:

1. Pengujian pertama hingga ketiga menggunakan 100 dataset (dataset kecil)
2. Pengujian keempat hingga keenam menggunakan 1.000 dataset (dataset sedang)
3. Pengujian ketujuh hingga kesembilan menggunakan 10.000 dataset (dataset besar)

Pengujian dilakukan tiga kali dalam setiap algoritma dengan dataset yang berbeda, hal ini bertujuan untuk menjamin hasil yang konsisten dan memperkuat keyakinan terhadap data yang diperoleh dari pengujian [21]. Selama proses pengujian, hanya 2 aplikasi saja yang digunakan karena bertujuan untuk memastikan bahwa tidak ada gangguan dari aplikasi lain yang dapat mempengaruhi kinerja CPU dan memori. Ke dua aplikasi tersebut adalah *CodeBlocks* untuk menjalankan algoritma untuk pengujian dan

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

Sniping Tools yang berfungsi untuk mengambil tangkapan layar hasil pengujian. Selanjutnya, setiap algoritma akan dijelaskan lebih rinci.

3.1. *Binary Searching Algorithm*

```
// Fungsi utama Binary Search dalam bahasa pemrograman C++
// untuk mencari elemen 'target' dalam vektor 'arr' yang sudah terurut secara menaik.
// Fungsi juga mencatat estimasi penggunaan memori selama proses pencarian.

int binarySearch(const vector<int>& arr, int target, size_t& memoryUsed) {

    int left = 0, right = arr.size() - 1;

    // Inisialisasi memori yang digunakan oleh variabel 'left' dan 'right'
    memoryUsed = sizeof(left) + sizeof(right);

    // Perulangan dilakukan selama indeks kiri tidak melebihi indeks kanan
    while (left <= right) {
        // Menghitung indeks tengah dengan formula aman dari overflow
        int mid = left + (right - left) / 2;

        // Tambahkan penggunaan memori untuk variabel 'mid'
        memoryUsed += sizeof(mid);

        // Tambahkan estimasi memori untuk elemen yang diakses (arr[mid])
        memoryUsed += sizeof(arr[mid]);

        // Jika elemen tengah adalah target, kembalikan indeksnya
        if (arr[mid] == target) {
            return mid;
        }
        // Jika elemen tengah lebih kecil dari target, pencarian dilanjutkan ke kanan
        else if (arr[mid] < target) {
            left = mid + 1;
        }
        // Jika elemen tengah lebih besar dari target, pencarian dilanjutkan ke kiri
        else {
            right = mid - 1;
        }
    }

    // Jika elemen tidak ditemukan, kembalikan -1 sebagai tanda tidak ditemukan
    return -1;
}
```

Kode program di atas merupakan implementasi algoritma *Binary Search* dalam bahasa pemrograman C++. Metode *Binary Search* sangat efisien untuk menemukan elemen tertentu dalam struktur data yang telah terurut secara menaik (*ascending*). kode ini menerima tiga parameter: vektor integer arr sebagai ruang pencarian, integer target sebagai nilai yang ingin ditemukan, dan variabel referensi *memoryUsed* yang mencatat estimasi penggunaan memori selama proses eksekusi. Pada awal fungsi, dua variabel yaitu *left* dan *right*, digunakan untuk menandai batas kiri dan kanan dari ruang pencarian. Estimasi awal memori yang digunakan dihitung berdasarkan alokasi kedua variabel ini.

Selama proses pencarian, dilakukan perulangan *(loop)* menggunakan struktur *while* yang terus berjalan selama indeks kiri *(left)* tidak melebihi indeks kanan *(right)*. Di setiap iterasi, ineks tengah *mid* dihitung menggunakan rumus *left + (right - left) / 2* untuk menghindari potensi *overflow* yang dapat terjadi jika menggunakan *left + right) / 2*. Nilai *mid* kemudian digunakan untuk mengakses elemen tengah dari

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

array. Estimasi memori ditambahkan dengan ukuran variabel *mid* serta ukuran dari elemen *arr[mid]* yang diakses, sebagai pendekatan kuantitatif terhadap penggunaan sumber daya memori. Proses perbandingan dilakukan untuk menentukan apakah *arr[mid]* sama dengan target, atau apakah pencarian harus dilanjutkan ke kiri atau ke kanan dari array. Jika elemen ditemukan, fungsi akan mengembalikan indeksnya, sedangkan jika tidak ditemukan setelah seluruh iterasi, fungsi akan mengembalikan nilai -1.

Dataset yang telah dibuat nantinya akan dimasukkan ke dalam program sesuai dengan alur pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. hasil pengujian dapat dilihat pada Tabel 2, sementara dokumentasi pengujian dapat dilihat di: https://bit.ly/4cSALcU.

Tabel 2. Hasil Pengujian Algoritma *Binary Search*

Pengujian ke	Jumlah data	Penggunaan Memori (byte)	Waktu Komputasi (detik)
1	100	64	0,072
2	100	40	0,013
3	100	64	0,018
4	1.000	80	0,072
5	1.000	88	0,018
6	1.000	88	0,016
7	10.000	112	0,077
8	10.000	112	0,013
9	10.000	112	0,031

Berdasarkan Tabel 2, jika dilihat dari segi penggunaan memori, rata-rata pengujian dengan 100 data adalah 56 byte, dengan 1.000 data adalah 85,3 byte, dan dengan 10.000 data adalah 112 byte. Namun dari sisi waktu komputasi, rata-rata waktu komputasi pada pengujian 100 data adalah 0,034 detik, dengan 1.000 data adalah 0,035 detik, dan dengan 10.000 data adalah 0,040 detik.

3.2. *Sequential Searching Algorithm*

```
// Fungsi utama Sequential Search dalam bahasa pemrograman C++
// untuk mencari elemen 'target' dalam vektor 'arr'.
// Fungsi ini juga mencatat estimasi penggunaan memori selama proses pencarian.

int sequentialSearch(const vector<int>& arr, int target, size_t& memoryUsed) {

    // Inisialisasi estimasi memori: memori untuk 'target' dan variabel penghitung indeks

    memoryUsed = sizeof(target) + sizeof(size_t);

    // Iterasi melalui seluruh elemen array dari indeks 0 hingga arr.size() - 1
    for (size_t i = 0; i < arr.size(); i++) {
        // Jika elemen pada indeks ke-i sama dengan target, kembalikan indeksnya
        if (arr[i] == target) {
            return i;
        }
    }

    // Jika elemen tidak ditemukan dalam array, kembalikan -1
    return -1;
}
```

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

Kode program di atas merupakan implementasi dari algoritma *Sequential Search* dalam bahasa pemrograman C++. Algoritma ini digunakan untuk mencari suatu elemen target dalam sebuah vektor *arr* yang tidak mensyaratkan adanya pengurutan sebelumnya. Prinsip kerja dari pencarian linier sangat sederhana, yaitu dengan memeriksa setiap elemen satu per satu mulai dari indeks pertama hingga akhir. Apabila elemen yang dicari ditemukan, maka indeks dari elemen tersebut langsung dikembalikan sebagai hasil. Jika elemen tidak ditemukan setelah seluruh elemen diperiksa, maka fungsi akan mengembalikan nilai -1, sebagai penanda bahwa elemen tidak ada dalam *array*.

Fungsi ini menerima tiga parameter, yakni *arr* sebagai vektor integer input, target sebagai nilai yang ingin dicari, dan *memoryUsed* sebagai referensi yang akan diisi dengan estimasi total penggunaan memori selama proses *search* berlangsung. Estimasi memori awal dihitung dari ukuran variabel target dan variabel penghitung indeks ***i*** yang digunakan dalam perulangan, karena tidak terdapat alokasi memori tambahan di dalam perulangan itu sendiri. Hal ini berbeda dengan algoritma lain seperti pencarian biner yang membutuhkan variabel tambahan seperti *left*, *right*, dan *mid*. Dari sisi kompleksitas waktu, algoritma pencarian linier memiliki performa ***O(n)***, di mana n adalah jumlah elemen dalam *array*. Hal ini disebabkan karena pada kasus terburuk *(worst-case)*, algoritma harus memeriksa seluruh elemen. Namun, keunggulan dari algoritma ini adalah kesederhanaan implementasi dan tidak adanya kebutuhan akan data yang terurut, sehingga sangat cocok digunakan dalam situasi di mana ukuran data kecil atau data tidak terstruktur.

Dataset yang telah dibuat nantinya akan dimasukkan ke dalam program sesuai dengan alur pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. hasil pengujian dapat dilihat pada Tabel 3, sementara dokumentasi pengujian dapat dilihat di: https://bit.ly/4cSALcU.

Tabel 3. Hasil Pengujian Algoritma *Sequential Search*

Pengujian ke	Jumlah data	Penggunaan Memori (byte)	Waktu Komputasi (detik)
1	100	12	0,016
2	100	12	0,020
3	100	12	0,022
4	1.000	12	0,017
5	1.000	12	0,020
6	1.000	12	0,022
7	10.000	12	0,018
8	10.000	12	0,023
9	10.000	12	0,013

Berdasarkan Tabel 3, jika dilihat dari segi penggunaan memori, rata-rata pengujian dengan 100, 1.000, dan 10.000 data memiliki rata-rata penggunaan memori yang sama yaitu 12 byte. Namun dari sisi waktu komputasi, rata-rata waktu komputasi pada pengujian 100 data adalah 0,019 detik, dengan 1.000 data adalah 0,020 detik, dan dengan 10.000 data adalah 0,018 detik.

3.3. *Interpolation Searching Algorithm*

```
// Fungsi utama Interpolation Search dalam bahasa pemrograman C++
// untuk mencari elemen 'target' dalam vektor 'arr' yang diasumsikan terurut secara menaik dan distribusinya merata.
// Fungsi ini juga mencatat estimasi penggunaan memori selama proses pencarian.

int interpolationSearch(const vector<int>& arr, int target, size_t& memoryUsed) {

    int low = 0, high = arr.size() - 1;

    // Estimasi awal penggunaan memori untuk variabel low dan high
    memoryUsed = sizeof(low) + sizeof(high);
```

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

```
    // Perulangan dilakukan selama batas pencarian valid dan target berada dalam rentang array
    while (low <= high && target >= arr[low] && target <= arr[high]) {
        // Jika posisi low dan high sama, cukup bandingkan langsung
        if (low == high) {
            if (arr[low] == target)
                return low;
            return -1;
        }

        // Menghitung posisi estimasi (pos) menggunakan rumus interpolasi
        int pos = low + ((double)(high - low) / (arr[high] - arr[low]) * (target - arr[low]));
        // Tambahkan estimasi penggunaan memori untuk variabel pos dan elemen arr[pos]
        memoryUsed += sizeof(pos) + sizeof(arr[pos]);

        // Jika elemen pada posisi pos adalah target, kembalikan pos
        if (arr[pos] == target) {
            return pos;
        }

        // Jika elemen pada pos lebih kecil dari target, pencarian lanjut ke kanan
        if (arr[pos] < target) {
            low = pos + 1;
        }
        // Jika elemen lebih besar, pencarian lanjut ke kiri
        else {
            high = pos - 1;
        }
    }

    // Jika elemen tidak ditemukan dalam rentang pencarian, kembalikan -1
    return -1;
}
```

Kode program di atas merupakan implementasi dari algoritma *Interpolation Search* menggunakan bahasa pemrograman C++, yang digunakan untuk mencari posisi elemen target dalam vektor *arr* yang sudah terurut secara menaik dan diasumsikan memiliki distribusi data yang merata. Berbeda dengan algoritma *Binary Search* yang selalu membagi ruang *search* secara simetris di tengah, *Interpolation Search* memanfaatkan rumus matematis untuk memperkirakan posisi elemen berdasarkan nilai relatifnya terhadap batas bawah *(low)* dan batas atas *(high)*. Estimasi posisi dihitung menggunakan formula *Interpolation*:

pos = low + ((target - arr[low]) × (high - low)) / (arr[high] - arr[low])

yang memungkinkan pengurangan jumlah iterasi secara signifikan pada distribusi data yang merata atau hampir linier. Fungsi ini dimulai dengan inisialisasi variabel *low* dan *high*, yang masing-masing menunjukkan indeks awal dan akhir dari vektor. Estimasi awal penggunaan memori dicatat berdasarkan ukuran kedua variabel ini. Selama kondisi *low* <= *high* terpenuhi dan target berada dalam rentang *arr[low]* hingga *arr[high]*, proses iteratif dilakukan. Jika low sama dengan high, maka cukup dilakukan pemeriksaan satu elemen. Posisi yang diperkirakan (pos) dihitung dengan rumus *Interpolation*, dan estimasi memori diperbarui dengan menambahkan ukuran variabel pos dan elemen *arr[pos]* yang diakses. Kemudian, algoritma membandingkan nilai *arr[pos]* dengan target. Jika sama, maka indeks pos dikembalikan. Jika nilai di posisi tersebut lebih kecil dari target, ruang pencarian digeser ke kanan *(low = pos + 1)*, dan sebaliknya jika lebih besar, pencarian digeser ke kiri *(high = pos - 1)*.

Dari segi kompleksitas waktu, algoritma ini memiliki performa rata-rata ***O(log log n)*** pada distribusi data yang merata, namun dapat memburuk menjadi ***O(n)*** jika data tidak terdistribusi dengan baik. Secara umum, algoritma *Interpolation Search* memberikan peningkatan efisiensi waktu dibandingkan *Binary Search* dalam kasus tertentu, namun dengan asumsi kuat bahwa nilai-nilai dalam array tersebar

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857

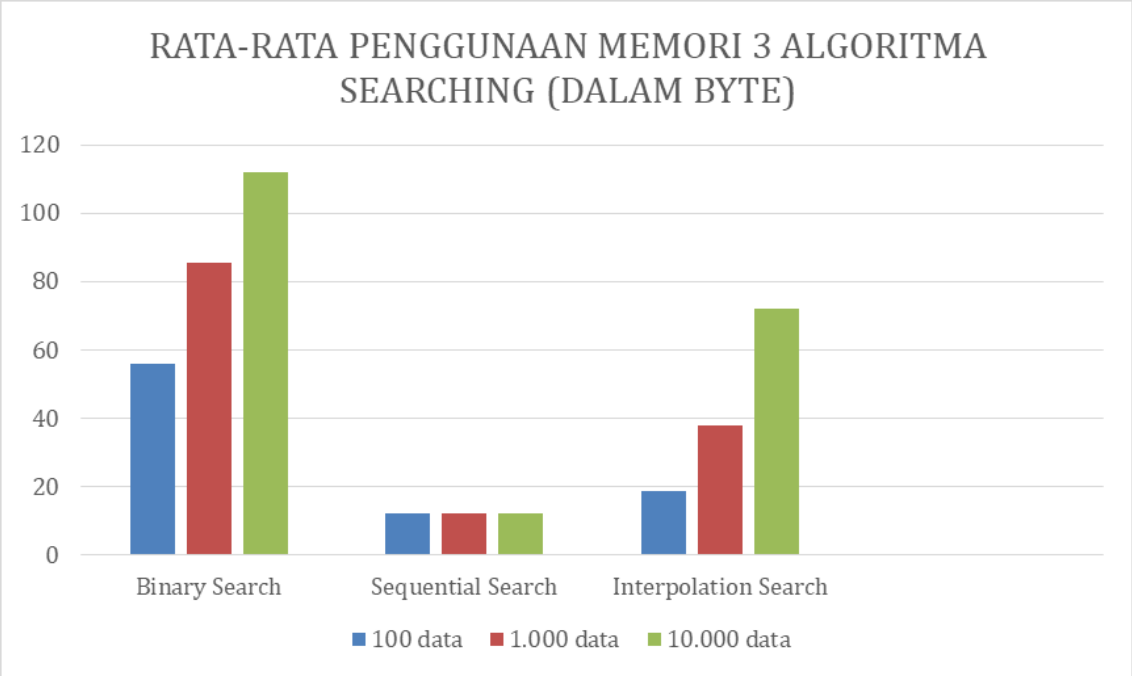
merata. Selain itu, fungsi ini juga mencatat estimasi penggunaan memori selama proses *search*, menjadikannya relevan dalam analisis kinerja algoritma yang mempertimbangkan baik efisiensi ruang maupun waktu. Oleh karena itu, algoritma ini memiliki keunggulan dalam skenario *search* yang melibatkan data numerik besar dengan distribusi yang seragam.

Dataset yang telah dibuat nantinya akan dimasukkan ke dalam program sesuai dengan alur pengujian yang telah ditentukan. Pengujian pertama hingga ketiga menggunakan dataset kecil dengan 100 data, pengujian keempat hingga keenam menggunakan dataset sedang dengan 1.000 data, dan pengujian ketujuh hingga kesembilan menggunakan dataset besar dengan 10.000 data. hasil pengujian dapat dilihat pada Tabel 4, sementara dokumentasi pengujian dapat dilihat di: https://bit.ly/4cSALcU.

Tabel 4. Hasil Pengujian Algoritma *Interpolation Search*

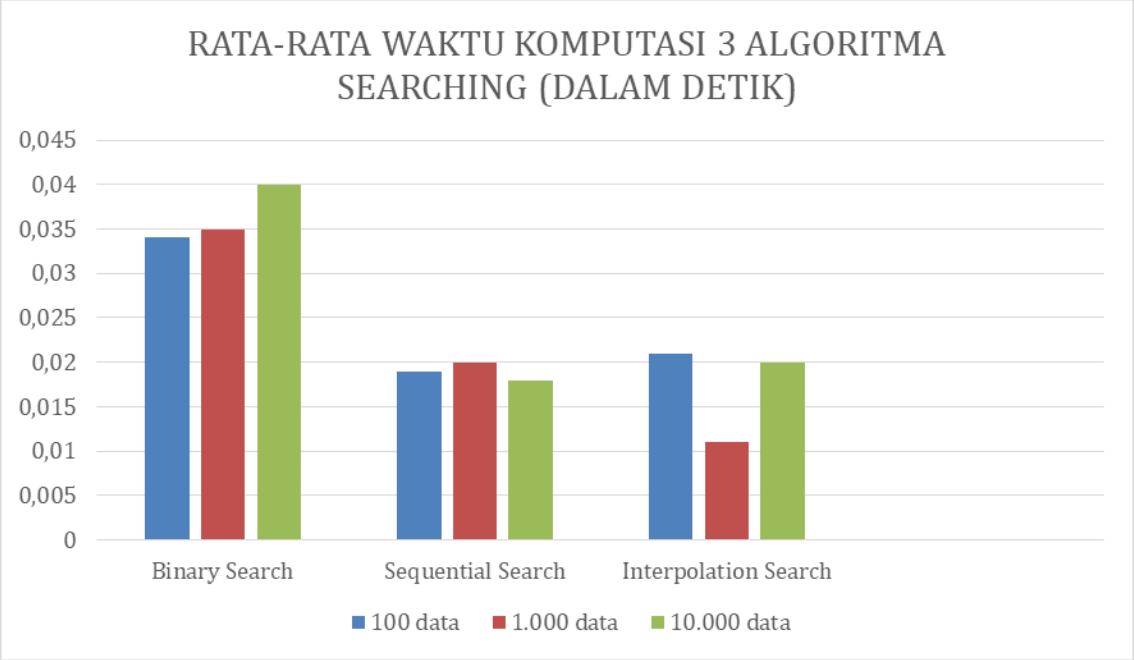
Pengujian ke	Jumlah data	Penggunaan Memori (byte)	Waktu Komputasi (detik)
1	100	16	0,021
2	100	24	0,020
3	100	16	0,023
4	1.000	40	0,009
5	1.000	40	0,010
6	1.000	32	0,014
7	10.000	64	0,022
8	10.000	80	0,020
9	10.000	72	0,017

Berdasarkan Tabel 4, jika dilihat dari segi penggunaan memori, rata-rata pengujian dengan 100 data adalah 18,7 byte, dengan 1.000 data adalah 37,3 byte, dan dengan 10.000 data adalah 72 byte. Namun dari sisi waktu komputasi, rata-rata waktu komputasi pada pengujian 100 data adalah 0,021 detik, dengan 1.000 data adalah 0,011 detik, dan dengan 10.000 data adalah 0,020 detik. Dari hasil pengujian ketiga algoritma yang telah dilakukan, rangkuman rata-rata hasil pengujian pada ketiga algoritma tersebut dapat dilihat pada Gambar 2 dan Gambar 3 di bawah ini.



Gambar 2. Grafik Rata-Rata Penggunaan Memori 3 Algoritma *Searching*

Jurnal Sistem Informasi dan Sistem Komputer, Vol. 10, No. 2, Juli 2025
ISSN: 2715-906X (Online)
10.51717/simkom.v10i2.857



Gambar 3. Grafik Rata-Rata Waktu Komputasi 3 Algoritma *Searching*

Berdasarkan Gambar 2 dan Gambar 3 yang menampilkan rata-rata hasil uji dari tiga algoritma pencarian untuk tiga ukuran data berbeda, yakni 100, 1.000, dan 10.000, diperoleh beberapa informasi sebagai berikut:

1. Pada data berjumlah 100, ketiga algoritma menunjukkan perbedaan dalam konsumsi memori. *Binary Search* mencatat penggunaan memori tertinggi dengan rata-rata yaitu 56 byte, disusul dengan *Interpolation Search* dengan rata-rata yaitu 18,7 byte, sementara *Sequential Search* menjadi yang paling efisien dalam hal ini dengan rata-rata yaitu 12 byte. Untuk waktu komputasi, *Interpolation Search* dengan rata-rata waktu komputasi yaitu 0,021 dan *Sequential Search* dengan rata-rata waktu komputasi yaitu 0,019 memiliki performa yang hampir sebanding dan lebih unggul kecepatannya dengan rata-rata waktu komputasi dibandingkan *Binary Search* yang memiliki rata-rata waktu komputasi yaitu 0,04 detik.
2. Saat jumlah data bertambah menjadi 1.000, *Binary Search* masih tercatat sebagai algoritma dengan konsumsi memori terbesar dengan rata-rata penggunaan memori yaitu 85,3 byte. Sebaliknya, *Sequential Search* tetap menjadi yang paling hemat dalam penggunaan memori dengan rata-rata yaitu 12 byte. Sementara itu, *Interpolation Search* menunjukkan kinerja terbaik dalam hal waktu komputasi, lebih cepat dari dua algoritma lainnya dengan rata-rata waktu komputasi yaitu 0,011 detik.
3. Pada skala data 10.000, penggunaan memori meningkat cukup signifikan untuk *Binary Search* dengan rata-rata yaitu 112 byte, disisi lain *Interpolation Search* yaitu dengan rata-rata 72 byte menjadi urutan yang kedua dalam hal penggunaan memori, *Sequential Search* tetap menjadi algoritma dengan penggunaan memori paling rendah dengan rata-rata yaitu 12 byte. Untuk kecepatan pencarian, *Sequential Search* dengan *Interpolation Search* memiliki performa yang hampir sebanding yaitu dengan rata-rata 0,018 dan 0,020 detik, diikuti oleh *Binary Search* sebagai waktu komputasi terlama yaitu dengan rata-rata 0,040 detik.

4. Kesimpulan dan Saran

4.1. Kesimpulan

Berdasarkan hasil uji terhadap tiga algoritma *search*, yaitu *Sequential Search*, *Binary Search*, dan *Interpolation Search*, pada tiga dataset yang berbeda (100, 1.000, dan 10.000 data), hasil yang diperoleh menunjukkan bahwa *Sequential Search* secara konsisten menjadi algoritma paling efisien dalam hal penggunaan memori, sementara *Binary Search* cenderung membutuhkan memori paling

besar, khususnya pada dataset besar (10.00 data). Dari sisi kecepatan eksekusi, *Interpolation Search* menunjukkan performa paling unggul pada dataset berukuran menengah (1.000 data), sedangkan *Sequential Search* tetap kompetitif, terutama pada data kecil (100 data) dan data besar (10.000 data). Temuan ini mengindikasikan bahwa pemilihan algoritma *search* harus mempertimbangkan prioritas sistem, apakah lebih menekankan pada efisiensi ruang atau waktu. Dalam konteks yang lebih luas, *Interpolation Search* dapat menjadi pilihan utama dalam aplikasi yang memerlukan efisiensi waktu, selama struktur dan sebaran data mendukung penerapannya.

4.2. Saran

Penelitian selanjutnya, agar dilakukan pengujian lebih lanjut dengan skala data yang lebih besar dan beragam distribusi nilai, seperti data yang bersifat acak, terurut menurun, maupun data dengan sebaran non-linear, guna mengevaluasi performa masing-masing algoritma secara lebih komprehensif. Selain itu, implementasi algoritma search dapat dikembangkan dalam berbagai lingkungan pemrograman atau platform yang berbeda, seperti Python, Java, atau C#, untuk mengamati pengaruh lingkungan eksekusi terhadap performa aktual algoritma, khususnya dalam hal manajemen memori dan efisiensi waktu.

5. Ucapan Terima Kasih

Kami mengucapkan terima kasih yang sebesar-besarnya kepada UIN K.H. Abdurrahman Wahid Pekalongan atas dukungan sarana dan prasarana yang telah disediakan selama proses penelitian ini berlangsung. Kehadiran lingkungan akademik yang kondusif serta akses terhadap berbagai sumber daya pembelajaran sangat membantu dalam menyusun dan menyelesaikan penelitian ini. Semoga hasil dari penelitian ini dapat memberikan kontribusi yang bermanfaat, baik dalam pengembangan ilmu pengetahuan maupun dalam praktik dunia nyata, khususnya di bidang algoritma dan pemrograman.

Daftar Pustaka

[1] A. dwi Aprilliani, “Analisis Algoritma Pencarian dalam Ilmu Komputer,” Sumedang, Nov. 2024.
[2] H. Maroli, T. Lase, J. Siregar, A. Metode, P. Linier, and D. Interpolasi, “Aplikasi Metode,” *Jurnal Teknologi Informasi dan Industri*, vol. 3, no. 1, 2023.
[3] T. Elizabeth, “Implementasi Algoritma Sequential Search Dan Binary Search Dalam Pencarian Data Faktur,” *Jurnal Teknik Informatika dan Sistem Informasi*, vol. 11, no. 2, 2024, [Online]. Available: http://jurnal.mdp.ac.id
[4] W. Kustiawan, “Pengamatan tentang optimalisasi algoritma pencarian dalam pemrosesan data besar.”
[5] H. Ramadhan and D. Avrilia Lantana, “Perbandingan Algoritma Binary Search dan Sequential Search untuk Pencarian Persediaan Stok Barang Berbasis Web.”
[6] R. Munir, *Algoritma & Pemrograman Bahasa Pascal dan C Edisi Revisi (Edisi Revisi)*. Bandung, 2011.
[7] H. Situmorang, “Analisa Algoritma pada Metoda Pencarian Linier, Biner dan Interpolasi,” *Jurnal Mahajana Informasi*, vol. 2, no. 2, 2017.
[8] L. Sitorus, *Algoritma dan Pemprograman*. Yogyakarta: CV. Andi Offset, 2015.
[9] A. Lin, “Binary Search Algorithm,” *WikiJournal of Science*, vol. 2, no. 1, 2019, doi: 10.15347/wjs/2019.005.
[10] A. Febryanto, “Penerapan Algoritma Sequential Search untuk Mencari Data Siswa Pada Sekolah Menengah Kejuruan Negeri 3 Bengkalis,” vol. 2, no. 1, pp. 51–59, 2022.
[11] A. M. Sajiah, V. O. Y. Ismail, Sutardi, and N. Ransi, “Implementasi Algoritma Bi-Linear Search untuk Pencarian Kode Buku Berbasis Web,” vol. 7, pp. 178–184, 2022.

[12] M. T. D. Putra, Munawir, and A. R. Yuniarti, *Belajar Pemprograman Lanjut dengan C++*. Bandung: Widina Media Utama, 2023.
[13] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms Third Edition."
[14] B. Jumadi *et al.*, "Implementasi Algoritma Binary Search pada Pencarian Data Jemaat Gereja HKBP Manado".
[15] D. E. Knuth, "The Art of Computer Programming," 2019. [Online]. Available: http://www-cs-faculty.stanford.edu/~knuth/taocp.html
[16] M. A. Fauzan, "Strategi Algoritma Algoritma Pencarian," 2022.
[17] K. Mehlhorn and P. Sanders, "Algorithms and Data Structures," 2007.
[18] W. L. Yuniar and F. Amin, "Sistem Pencarian Naskah Dinas pada Polres Kendal dengan Algoritma Sequential Search," *Jurnal Manajemen informatika & Sistem Informasi)*, vol. 4, no. 2, 2021, [Online]. Available: http://e-journal.stmiklombok.ac.id/index.php/misi
[19] D. Markuci and C. Prianto, "Analisis Perbandingan Penggunaan Algoritma Sequential Search Dan Binary Search Pada Aplikasi Surat Perjalanan Dinas," vol. 6, pp. 110–119, 2022.
[20] Rahmaddeni, "Analisa Perbandingan Algoritma Pencarian (Searching Algoritm)," vol. 1, 2012.
[21] I. P. Pujiono, R. B. Trianto, and F. M. Hana, "Perbandingan Efisiensi Memori dan Waktu Komputasi Pada 7 Algoritma Sorting Menggunakan Bahasa Pemrograman Java," *Simkom*, vol. 9, no. 2, pp. 218–230, Jul. 2024, doi: 10.51717/simkom.v9i2.481.