Komputika: Jurnal Sistem Komputer
Volume 11, Nomor 2, Oktober 2022, hlm. 143 - 150
Terakreditasi Peringkat 3, SK No. 28/E/KPT/2019
DOI: **10.34010/komputika.v11i2.5493**

ISSN: 2252-9039 (print)
ISSN: 2655-3198 (online)

Analisis Perbandingan Kinerja Metode Rekursif dan Metode Iteratif dalam Algoritma Linear Search

Erba Lutfina[1*], Nur Inayati[2], Galuh Wilujeng Saraswati[3]

[1)]Program Studi Sistem Informasi, Fakultas Sains dan Teknologi, Universitas Nasional Karangturi
Jl. Raden Patah No. 182 – 192, Semarang, Indonesia 50127

[2))]Program Studi Sistem Informasi, Fakultas Sains dan Teknologi, Universitas Terbuka
Jl. Pantura Semarang - Kendal No.Km 14, Semarang, Indonesia 50156

[3))]Program Studi Teknik Informatika, Fakultas Ilmu Komputer, Universitas Dian Nuswantoro
Jl. Imam Bonjol No.207, Semarang, Indonesia 50131

**email: erbalutfina@gmail.com*

(Naskah masuk: *4 Oktober 2021*; diterima untuk diterbitkan: *14 Februari 2022*)

ABSTRAK *– Salah satu algoritma pencarian data yang paling populer adalah algoritma linear search. Dalam proses pencarian data sebuah list menggunakan algoritma linear search dapat diterapkan dengan cara iteratif dan rekursif. Pandangan umum mengenai algoritma linear search adalah bahwa performa metode iteratif memiliki hasil yang sama dengan rekursif. Namun di beberapa penelitian menentang pernyataan tersebut yang mungkin tidak berlaku pada semua kasus. Dari analisis tersebut, penelitian ini berfokus pada perbandingan metode rekursif dan iteratif pada algoritma linear search untuk mengetahui algoritma mana yang paling sesuai, efisien dan efektif. Penelitian dilakukan menggunakan 3 studi kasus dengan masing-masing data sebanyak 1 juta, 10 juta, dan 100 juta. Penelitian berfokus pada hasil penggunaan memori dan waktu akses pada proses pencarian data menggunakan notasi Big-O dan bahasa pemrograman Python. Hasil penelitian menunjukkan bahwa algoritma linear search secara iteratif lebih efektif dan efisien dari pada rekursif, meskipun kedua metode tersebut memiliki kompleksitas Big-O yang sama. Dengan hasil algoritma rekursif memiliki hasil waktu eksekusi sebesar 25.5147 s, 26.847 s, dan 28.3863 s serta 58.3 MiB, 406.9 MiB, dan 3882.3 MiB untuk penggunaan memori. Sedangkan algoritma iteratif hanya membutuhkan waktu eksekusi sebesar sebesar 0.0143 s, 0.016 s, dan 0.0133 s serta 57.2 MiB, 405.1 MiB, dan 3881.7 MiB untuk penggunaan memori.*

Kata Kunci *– Searching; Linear Search; Iteratif; Rekursif; Data.*

A Comparative Analysis of the Performance of Recursive Methods and Iterative Methods in Linear Search Algorithms

ABSTRACT *–The linear search algorithm is one of the most popular data search algorithms. In the process of searching data for a list using a linear search algorithm, it can be applied in an iterative and recursive way. The general view of linear search algorithms is that the iterative methods perform the same as recursive methods. However, some studies contradict this statement which may not apply in all cases. From this analysis, this study focuses on the comparison of recursive and iterative methods on linear search algorithms to find out which algorithm is the most suitable, efficient, and effective. The research was conducted using 3 case studies with data of 1 million, 10 million, and 100 million respectively. The research focuses on the results of memory usage and access time in the data search process using Big-O notation and Python programming language. The results show that the iterative algorithm is more effective and efficient than recursive, although both methods have the same Big-O complexity. With the results of the linear search algorithm recursively, the execution time results are 25.5147 s, 26.847 s, and 28.3863 s and 58.3 MiB, 406.9 MiB, and 3882.3 MiB for memory usage. While the iterative algorithm only requires execution time of 0.0143 s, 0.016 s, and 0.0133 s as well as 57.2 MiB, 405.1 MiB, and 3881.7 MiB for memory usage.*

Keywords *- Searching; Linear Search; Iteratif; Rekursif; Data.*

1. PENDAHULUAN

Pencarian data dalam sebuah list merupakan aspek dasar dalam dunia komputasi [1]. Terdapat sejumlah algoritma yang telah dikembangkan untuk menangani proses pencarian data [2]. Salah satu algoritma pencarian data yang paling populer adalah algoritma linear search [3]. Algoritma *linear search* memindai elemen-elemen dalam list secara linier atau berurutan dengan memindai satu per satu elemen dalam list [4]. Dalam proses pencarian data sebuah list menggunakan algoritma *linear search* dapat diterapkan dengan cara iteratif dan rekursif [5]. Kedua algoritma tersebut memiliki perbedaan mengenai aspek efisiensi performa pada program [6]. Untuk mengetahui algoritma mana yang lebih efektif dan efisien terdapat beberapa solusi salah satunya dengan membandingkan kedua algoritma [7]. Perbandingan algoritma dilakukan dengan menganalisis performa dari algoritma sehingga didapatkan algoritma yang paling efektif dan efisien [8]. Analisis algoritma pada penelitian ini berfokus pada analisis performa dan kompleksitas dari kedua algoritma seperti waktu akses, penggunaan memori, serta notasi *Big-O* [9].

Terdapat beberapa peneliti yang berfokus pada perbandingan pemrograman dengan teknik rekursive dan iterative [10]. Seperti pada penelitian Kunkle dan Allen [11] yang menyelidiki hubungan antara penulisan kode rekursif dan penulisan kode iteratif. Penelitian dilakukan menggunakan bahasa yang dirancang khusus untuk penelitian. Di dalam penelitian ditemukan transfer pengetahuan yang baik dari menulis fungsi iteratif ke menulis fungsi rekursif, tetapi tidak sebaliknya. Pada penelitian McCauley [12] mengemukakan bahwa iterasi memfasilitasi proses rekursi. Subjek yang digunakan pada penelitian tersebut tidak menggunakan pemrograman dan tidak membaca coding namun disajikan dengan spesifikasi tentang bagaimana cara mmperhitungkan solusi (secara iteratif atau rekursif). Mirolo [13] melakukan penelitian untuk menentukan metode manakah yang lebih mudah dipelajari, rekursi atau iterasi. Penelitian tersebut dilakukan berbasis kuesioner yang membutuhkan jawaban tertulis serta dengan menggunakan *coding*. Pada penelitan ini ditemukan bahwa mahasiswa lebih berhasil memecahkan masalah dengan iterasi daripada dengan rekursi. Namun tidak dijabarkan pertanyaan yang digunakan dalam penelitian dan tidak dijelaskan berapa banyak kode yang ditulis oleh koresponden. Pada penelitian Rinderknecht [14] mengemukakan bahwa subjek penelitian mampu mentransfer pengalaman dari metode iterasi ke rekursi serta adanya pengaruh timbal balik yang kompleks pada metode iterasi dan rekursi. Namun, peneliti tidak dapat mengidentifikasi pengaruh tersebut secara detail. Pada penelitian Zhang [15] dilakukan perbandingan dan analisis untuk berbagai masalah antara sirkuit FPGA yang diimplementasikan secara iteratif dan rekursif. Penelitian dilakukan dengan melakukan eksperiman pada empat kasus yang berbeda. Dalam penelitian digunakan dua tipe metode rekursif yaitu *linear search* dan *binary search*. Hasil penelitian menunjukkan bahwa dari pengujian secara acak 9 contoh eksperimen didapatkan bahwa algoritma rekursif dapat meningkatkan sumber daya hardware dan waktu eksekusi yang dibutuhkan. Namun pada sebagian besar eksperimen didapatkan bahwa metode iteratif memiliki hasil yang lebih unggul daripada rekursif.

Dari beberapa analisis penelitan yang telah dilakukan sebelumnya, untuk mencari algoritma mana yang lebih efisien dan efektif serta cocok pada data yang digunakan adalah dengan melakukan perbandingan algoritma [16]. Dari analisis tersebut, penelitian ini berfokus pada perbandingan metode rekursif dan iteratif pada algoritma *linear search* untuk mengetahui algoritma mana yang paling sesuai, efisien dan efektif. Penelitian ini bertujuan untuk menganalisis kinerja metode iteratif dan rekursif pada algoritma *linear search*. Perbandingan kedua metode berfokus pada hasil penggunaan memori dan waktu akses pada proses pencarian data menggunakan bahasa pemrograman Python.

2. METODE DAN BAHAN

Dataset

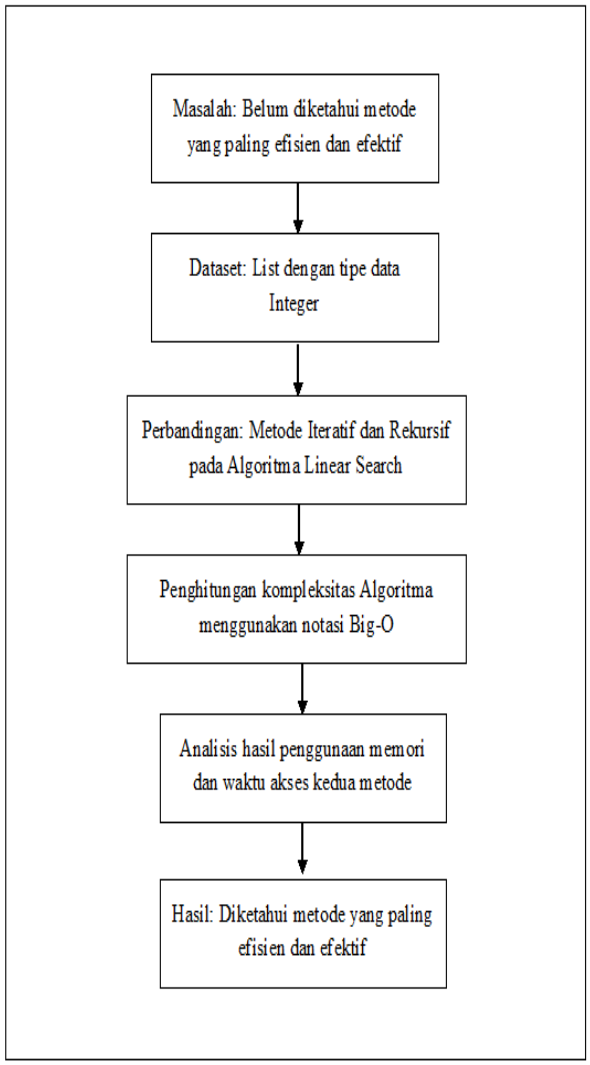
Dataset yang digunakan dalam penelitian ini adalah data dalam bentuk *list* dengan tipe data Integer. *List* tersebut digunakan sebagai parameter pada penerapan metode iteratif dan rekursif fungsi linear search. Nilai dari data *list* dihasilkan dengan cara menghasilkan data baru secara random. Terdapat 3 studi kasus dengan masing-masing data sebanyak 1 juta, 10 juta, dan 100 juta data dengan tiga data untuk proses pencarian

Strategi Penelitian

Penelitian ini menggunakan metode kualitatif sebagai acuan atau tolak ukur untuk menemukan dasar pengetahuan. Penelitian yang dilakukan bermaksud untuk menyelesaikan masalah yang ada, kemudian hasilnya digunakan sebagai acuan, pembanding serta sebagai tolak ukur. Uraian langkah-langkah dari metode yang diusulkan dapat dilihat pada gambar 1.

Dalam penelitian, proses implementasi dilakukan dengan menghitung kompleksitas algoritma iteratif dan rekursif pada *linear search* menggunakan bahasa pemrograman *python*. Tujuan dari penelitian ini adalah membandingkan hasil penghitungan

kompleksitas algoritma serta perbandingan performa dari tiap algoritma, sehingga diketahui algoritma mana yang paling efisien dan efektif. Proses implementasi dibagi menjadi dua tahapan yaitu implementasi pada algoritma *linear search* menggunakan metode iteratif dan rekursif. Tahapan selanjutnya adalah membandingkan performa yang dihasilkan seperti waktu akses dan pengguanaan memori menggunakan notasi *Big-O*.



Gambar 1. Strategi Penelitian

3. HASIL DAN PEMBAHASAN

Linear Search secara Iteratif

Penjabaran hasil dan pembahasan penelitian diawali dengan pembahasan penggunaan metode iteratif pada algoritma *linear search.* Pada penelitian ini algoritma *linear search* secara iteratif digunakan dalam proses pencarian item di dalam *list*. Bahasa pemrograman yang digunakan adalah *Python*. *Pseudocode* atau deskripsi dari algoritma *linear search* secara iteratif dapat dilihat pada tabel 1.

Tabel 1. *Pseudocode* algoritma *Linear Search Iterative*

LinearSearchIterative
Inputan A[0..N-1], TargetValue
TargetIndex ← -1 **for** each value in list **if** IndexValue = TargetValue **then** TargetIndex = IndexValue **break** **end if** **end for** return TargetIndex

Penjelasan dari *pseudocode* pada tabel 1 adalah sebagai berikut :

- Fungsi *LinearSearchIterative* memiliki argumen *List* dan *TargetValue.*
- Variabel *TargetIndex* diset ke -1.
- *For* dibuat dan melakukan iterasi pada setiap *value* di dalam list.
- Jika value memiliki nilai yang sama dengan *TargetValue,* maka nilai dari *TargetIndex* adalah *IndexValue.* Kemudian break berfungsi untuk menghentikan proses perulangan.
- Jika *value* tidak ditemukan maka perulangan akan berhenti dan akan mengembalikan TargetIndex bernilai -1 yang berarti fungsi tidak dapat menemukan elemen yang dicari.

Untuk mengetahui kompleksitas waktu dari algoritma *linear search iterative* dapat dilakukan dengan notasi *Big-O*. Tahapan-tahapan yang harus dianalisis adalah berapa banyak operasi yang akan dilakukan oleh fungsi *LinearSearchIterative*. Fungsi dijalankan dengan memeriksa semua elemen/value dari *List.* Jika *value* tersebut memiliki nilai yang sama dengan *TargetValue*, fungsi kemudian melakukan *assignment* selanjutnya.

Tahapan pertama adalah melakukan pengecekan pada line pertama yaitu TargetIndex ← -1. *Line* tersebut merupakan sebuah *operation*. Pengecekan kedua pada kondisi perulangan *for*, yang merupakan sebuah *loop* berukuran panjangnya *list*. Kemudian pada kondisi *if* dan statement didalamnya merupakan 3 operations yang berada dalam *loop*. Dari analisis tersebut terdapat tiga kasus yang dapat terjadi yaitu kasus terbaik, rata-rata, dan terburuk. Penjelasannya adalah sebagai berikut:

- Kasus terbaik terjadi ketika nilai elemen sama dengan elemen pertama dari *list*, dimana hanya diperlukan satu perbandingan. Kompleksitas waktu terbaik dari algoritma *linear search* iterative adalah O(n).

- Kasus rata-rata atau average terjadi ketika elemen yang dicari berada dalam inputan list yang besar. Dibutuhkan rata-rata perbandingan sebesar n/2. Waktu eksekusi algoritma sebanding dengan ukuran list. Jika ukuran list digandakan, maka jumlah perbandingan menjadi dua kali lipat. Dalam notasi *Big-O* konstanta perkalian ½ dapat dihilangkan karena konstanta perkalian tidak berdampak pada tingkat pertumbuhan (growth rate). Sehingga kompleksitas pada average case adalah O(n/2) atau O(n).
- Kasus terburuk terjadi ketika value tidak terdapat dalam list atau muncul di akhir list. Ketika kondisi ini terjadi maka algoritma akan membutuhkan perbandingan sebesar n.
  Misalkan $T_1(n)$, $T_2(n)$,… merupakan waktu eksekusi untuk semua kemungkinan input berukuran n. Maka kompleksitas waktu kasus terburuk $T(n) = \max(T_1(n), T_2(n), \ldots)$. Kompleksitas waktu kasus terburuk untuk algoritma linear search iterative menjadi T(n) = n atau O(n).

Dari *pseudocode* algoritma *Linear Search* Iterative di atas dapat diimplementasikan ke dalam *source code* pada gambar 2.

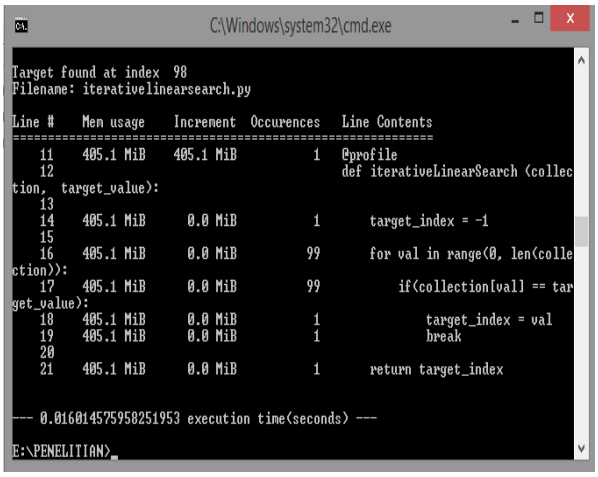
```
from memory_profiler import profile
import time

@profile
def iterativeLS (col, targetVal):

    targetIndex = -1

    for val in range(0, len(col)):
        if(col[val] == targetVal):
            targetIndex = val
            break

    return targetIndex

col = list(range(1, 100000000))
targetVal = 99

targetIndex = iterativeLS(col, targetVal)

if targetIndex != -1:
    print("Target found at index % d" % targetIndex)
else:
    print("Target not found in collection")

startTime = time.time()
iterativeLS(col, targetVal)
print("%sexecution time" %(time.time() - startTime))
```

Gambar 2. *Source code Linear Search* secara iteratif

Dari *source code* gambar 2, penjelasan tiap baris kode adalah sebagai berikut:

- Pertama, untuk mengetahui detail *runtime* dan penggunaan memori pada python dapat menggunakan Python *memory profiler*. Untuk menggunakan Python *memory profiler* harus melakukan instalasi melalui *package management system* yaitu pip
- Baris berikutnya dibuat definisi fungsi Linear Search secara iteratif dengan nama fungsi iterativeLS. Fungsi iterativeLS memiliki dua parameter yaitu col dan targetVal. Dimana col adalah koleksi array yang akan dicari dalam proses. Sedangkan targetVal adalah nilai yang akan dicari dalam proses
- Selanjutnya pengembalian nilai diatur secara default dengan menggunakan kode -1
- Pada baris selanjutnya yaitu perulangan secara iteratif atau biasa disebut dengan *iterative loop*. *Iterative loop* dimulai dari 0 dan berulang sampai sebelum nilai panjang dari seluruh koleksi (col). Panjang dari koleksi item col ditunjukkan dengan kode len(col). Indeks atau nomor iterasi saat ini (*current iteration*) disimpan di dalam variabel val
- Selanjutnya, *current value* pada index yang diberikan dibandingkan dengan nilai target. Jika cocok, nilai targetIndex akan diatur menjadi nilai iterasi saat ini dan keluar dari *loop*.
- Setelah mencapai *line* akhir dari fungsi iterativeLS, nilai targetIndex kemudian di-*return*. Pada tahap ini, nilai targetIndex dapat berupa nilai *default* atau index dari nilai yang ditemukan
- Keluar dari fungsi iterativeLS, nilai col dapat ditentukan nilainya secara bebas dengan menggunakan fungsi *list*
- Nilai targetVal juga dapat ditentukan secara bebas
- Tahapan untuk mengetahui *runtime* dan dapat menggunakan fungsi Python time.time(). Metode time.time() digunakan untuk mendapatkan waktu akses dalam detik sejak *epoch*. *Epoch* adalah titik di mana waktu dimulai, dan berbeda pada beberapa platform. Pada Windows dan sebagian besar sistem Unix, *epoch* dimulai dari 1 Januari 1970 00:00:00 (UTC) dan *leap seconds* tidak dihitung terhadap waktu dalam detik sejak *epoch*. Untuk memeriksa *epoch* pada platform tertentu, dapat menggunakan fungsi time.gmtime(0).
- Hasil dari *source code linear search* secara iteratif, contohnya dapat dilihat pada gambar 3. Dalam gambar 3 hasil *source code* diatas, runtime dan penggunaan memori sebuah proses diketahui dari pengaksesan file

berekstensi ".py" pada terminal atau *command prompt*.



Gambar 3. Hasil *runtime* dan penggunaan memori *Linear Search* secara iteratif

Linear Search secara Rekursif

Selanjutnya pembahasan algoritma *linear search* secara rekursif yang digunakan dalam proses pencarian item di dalam list. *Pseudocode* atau deskripsi dari algoritma *linear search* secara rekursif dapat dilihat pada tabel 2.

Tabel 2. *Pseudocode* algoritma *Linear Search Recursive*

LinearSearchRecursive
Inputan A[0..N-1], index, TargetValue

TargetIndex ← -1
if index < length of list
if first item of list = TargetValue
then return index of the item
else
return LinearSearchRecursive (A, remaining index, TargetValue)
else
 	
return TargetIndex
end if

Penjelasan *pseudocode* pada tabel 2 adalah sebagai berikut :

- Fungsi *LinearSearchRecursive* memiliki argumen *List, index* atau lokasi dari item *list*, dan *TargetValue.* Fungsi mencari value dalam rentang A[0..N-1].
- *List* adalah item dari array yang akan dicari.
- *Index* adalah posisi atau lokasi saat ini dari item.
- *TargetValue* adalah nilai yang sedang dicari.
- Variabel *TargetIndex* diset ke -1.
- Jika index dari *value* memiliki nilai kurang dari panjang dari *list,* maka dilakukan pengecekan kembali apakah item saat ini sama dengan nilai dari *TargetValue.* Jika memenuhi kondisi maka fungsi akan mengembalikan nilai index dari item. Jika tidak memenuhi maka *LinearSearchRecursive* dipanggil kembali.
- Kemudian jika index telah melampaui panjang dari list akan mengembalikan TargetIndex bernilai -1 yang berarti fungsi tidak dapat menemukan elemen yang dicari.

Untuk mengetahui kompleksitas waktu dari algoritma *linear search recursive* dilakukan dengan notasi *Big-O*. Tahapan-tahapan yang harus dianalisis adalah berapa banyak operasi yang akan dilakukan oleh fungsi *LinearSearchRecursive*. Fungsi dijalankan dengan memeriksa kondisi index dari *value* memiliki nilai kurang dari panjang dari *list.* Jika memenuhi kondisi maka dilakukan pengecekan kembali apakah item saat ini sama dengan nilai dari TargetValue, fungsi kemudian melakukan *assignment* selanjutnya. Jika tidak maka fungsi *LinearSearchRecursive* dipanggil kembali.

Tahapan pertama adalah melakukan pengecekan pada line pertama yaitu TargetIndex ← -1. Line tersebut merupakan sebuah *operation*. Pengecekan kedua pada kondisi if dan statement didalamnya merupakan 2 perbandingan. Pengecekan kedua kondisi *return* index dan *return LinearSearchRecursive* yang bernilai n/2 perbandingan. Dari analisis tersebut terdapat tiga kasus yang dapat terjadi yaitu kasus terbaik, rata-rata, dan terburuk. Penjelasannya adalah sebagai berikut:

- Kasus terbaik terjadi ketika nilai elemen sama dengan elemen pertama dari *list*, dimana hanya diperlukan satu perbandingan. Kompleksitas waktu terbaik dari algoritma linear search iterative adalah O(n).
- Kasus rata-rata atau *average* terjadi ketika elemen yang dicari berada dalam inputan list yang besar.
  Dibutuhkan rata-rata perbandingan sebesar n/2.
  Waktu eksekusi algoritma sebanding dengan ukuran list. Jika ukuran list digandakan, maka jumlah perbandingan menjadi dua kali lipat.
  Dalam notasi *Big-O* konstanta perkalian ½ dapat dihilangkan karena konstanta perkalian tidak berdampak pada tingkat pertumbuhan (growth rate).
  Sehingga kompleksitas pada *average case* adalah O(n/2) atau O(n).
- Kasus terburuk terjadi ketika *value* tidak terdapat dalam list atau muncul di akhir *list*. Ketika kondisi ini terjadi maka algoritma akan membutuhkan perbandingan sebesar n.

Misalkan $T_1(n)$, $T_2(n)$,… merupakan waktu eksekusi untuk semua kemungkinan input berukuran n.
Maka kompleksitas waktu kasus terburuk T(n) = max($T_1(n)$, $T_2(n)$, …). Kompleksitas waktu kasus terburuk untuk algoritma linear search iterative menjadi T(n) = n atau O(n). Terdapat perbedaan kasus terburuk pada metode rekursif dengan iteratif. Meskipun sama-sama bernilai O(n), metode rekursif akan cenderung mengisi *stack* jika *list* terlalu besar. Sehingga hasil waktu akses yang dihasilkan cenderung lebih besar.

Dari *pseudocode* algoritma *Linear Search* Recursive di atas dapat diimplementasikan ke dalam *source code* dapat dilihat pada gambar 4.

```
from memory_profiler import profile
import time

@profile
def recursiveLS (col, index, targetVal):
    if(index < len(col)):
        if(col[index] == targetVal):
            return index
        else:
            return recursiveLS(col, index + 1, targetVal)
    else:
        return -1

col = list(range(1, 100000000))
targetVal = 99

targetIndex = recursiveLS(col, 0, targetVal)

if target_index != -1:
    print("Target found at index % d" % targetIndex)
else:
    print("Target not found in collection")

startTime = time.time()
recursiveLS(col, 0, targetVal)
print("%s execution time " % (time.time() - startTime))
```

Gambar 4. *Source code Linear Search* secara rekursif

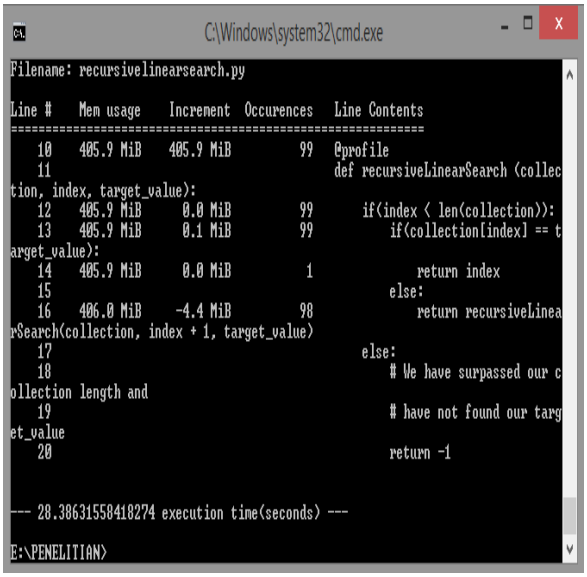
Dari *source code* diatas, penjelasan tiap baris kode adalah sebagai berikut:

- Pertama, untuk mengetahui detail *runtime* dan penggunaan memori pada python dapat menggunakan Python *memory profiler*. Untuk menggunakan Python *memory profiler* harus melakukan instalasi melalui *package management system* yaitu pip
- Baris berikutnya dibuat definisi fungsi *Linear Search* secara rekursif dengan nama fungsi *recursiveLS*. Fungsi *recursiveLS* memiliki tiga parameter yaitu col, index, dan targetVal. Dimana col adalah koleksi array yang akan dicari dalam proses. Variabel Index adalah indeks saat ini yang ada dalam proses pencarian. Sedangkan targetVal adalah nilai yang akan dicari dalam proses
- Selanjutnya dilakukan pengecekan kondisi *base case* (kasus dasar). Kondisi dasar akan memberi sinyal bahwa proses telah mencapai akhir dari rekursi. Setelah pengulangan pada semua elemen yang ada di variabel col, nilai akan bernilai *empty* (kosong) atau nilai indeks dari nilai target (targetVal)
- Pada baris selanjutnya yang ditunjukkan dengan kode if(col[index] == targetVal. Jika elemen yang dieksekusi belum habis, kode yang ada di dalam *loop* dieksekusi dan diperiksa *current value*-nya. Jika nilai *current value* adalah nilai target (targetVal), maka fungsi akan mengembalikan atau *return* nilai index.
- Di sisi lain, jika nilai target (targetVal) belum ditemukan pada *current index*, fungsi melakukan rekursi dengan memanggil dirinya sendiri dan meningkatkan indeks dengan menambahkan nilai 1. Kemudian proses menyerahkan jawaban untuk panggilan rekursif ke fungsi *recursiveLS* itu sendiri.
- Pada akhir fungsi recursiveLS, *base case* diselesaikan dengan mengembalikan nilai -1 jika *current index* melebihi jumlah elemen dalam koleksi col
- Keluar dari fungsi *recursiveLS*, nilai col dapat ditentukan nilainya secara bebas dengan menggunakan fungsi *list*
- Nilai targetVal juga dapat ditentukan secara bebas
- Tahapan untuk mengetahui *runtime* dan dapat menggunakan fungsi Python time.time(). Metode time.time() digunakan untuk mendapatkan waktu akses dalam detik sejak *epoch*. *Epoch* adalah titik di mana waktu dimulai, dan berbeda pada beberapa platform. Pada Windows dan sebagian besar sistem Unix, *epoch* dimulai dari 1 Januari 1970 00:00:00 (UTC) dan leap seconds tidak dihitung terhadap waktu dalam detik sejak *epoch*. Untuk memeriksa *epoch* pada platform tertentu, dapat menggunakan fungsi time.gmtime(0).
- Hasil dari *source code* linear search secara rekursif, contohnya dapat dilihat pada gambar 5. Dalam gambar 5 hasil *source code* diatas, *runtime* dan penggunaan memori sebuah proses diketahui dari pengaksesan file berekstensi ".py" pada terminal atau

command prompt.

Hubungan Kompleksitas Waktu dengan Waktu Akses Eksekusi Program

Pengujian terhadap algoritma *linear search* secara rekursif dan iteratif dilakukan untuk menemukan perbedaan hasil waktu akses dan besarnya memori yang digunakan. Proses eksekusi kedua algoritma menggunakan tiga kasus dengan masing-masing data list sebesar 1juta, 10 juta, dan 100 juta data. Data-data tersebut digunakan untuk mengeksekusi pencarian suatu data di dalam sebuah list. Perbandingan hasil dari algoritma *linear search* secara rekursif dan iteratif ditunjukkan oleh tabel 3.



Gambar 5. Hasil *runtime* dan penggunaan memori *Linear Search* secara rekursif

Tabel 3. Perbandingan hasil dari algoritma linear search secara rekursif dan iteratif

Data		Penggunaan Memori	Waktu Proses
1 juta data	iteratif	57.2 MiB	0.0143 s
	rekursif	58.3 MiB	25.5147 s
10 juta data	iteratif	405.1 MiB	0.016 s
	rekursif	406.9 MiB	26.847 s
100 juta data	iteratif	3881.7 MiB	0.0133 s
	rekursif	3882.3 MiB	28.3863 s

Algoritma *Linear Search* secara Iteratif dan Rekursif memiliki kompleksitas penggunaan ruang memori yang sama yaitu O(1) pada kasus terbaik dan O(n) pada kasus *average* dan terburuk. Namun dari analisis tabel diatas, kedua metode memiliki perbedaan hasil penggunaan memori yang berbeda. Algoritma linear search secara rekursif memiliki hasil secara berurutan sebesar 58.3 MiB, 406.9 MiB, dan 3882.3 MiB. Sedangkan untuk hasil penggunaan memori secara iteratif menunjukkan hasil yang lebih baik. Dengan hasil secara berurutan sebesar 57.2 MiB, 405.1 MiB, dan 3881.7 MiB. Meskipun dari hasil analisis notasi Big-O kedua algoritma memiliki hasil yang sama, namun algoritma linear search secara rekursif menggunakan memori yang lebih besar karena adanya proses *call stack*.

Sedangkan pada perbandingan hasil waktu akses kedua metode memiliki perbedaan yang cukup mencolok. Dari hasil tabel diatas waktu akses yang dibutuhkan oleh algoritma *linear search* secara rekursif memiliki hasil secara berurutan sebesar 25.5147 s, 26.847 s, dan 28.3863 s. Sedangkan waktu akses yang diperlukan oleh algoritma linear search secara iteratif secara berurutan adalah sebesar 0.0143 s, 0.016 s, dan 0.0133 s. Dari hasil tersebut metode iteratif menghasilkan waktu akses yang lebih cepat dari rekursif. Meskipun sama-sama bernilai O(n), metode rekursif akan cenderung mengisi stack jika list terlalu besar. Sehingga hasil waktu akses yang dihasilkan cenderung lebih besar.

4. KESIMPULAN

Dari hasil penelitian yang telah dilakukan, diketahui bahwa algoritma linear search secara iteratif lebih efektif dan efisien dari pada rekursif. Meskipun kedua metode tersebut memiliki kompleksitas *Big-O* yang sama, namun hasil dari eksekusi program menunjukkan hasil yang berbeda. Dari percobaan yang dilakukan algoritma linear search secara iteratif memiliki hasil waktu eksekusi dan penggunaan memori yang lebih unggul yaitu waktu akses dan penggunaan memori yang lebih sedikit dibanding metode rekursif. Hal ini terjadi karena metode rekursif menggunakan manipulasi *call stack* ketika list menjadi semakin besar. Untuk pengembangan penelitian kedepan, penelitian dapat dilakukan dengan menggunakan bahasa pemrograman lain serta dilakukan penambahan algoritma yang sudah ada untuk mengetahui algoritma mana yang paling efektif dan efisien.

DAFTAR PUSTAKA

[1] K. Roopa and J. Resshma, "A Comparative Study of Sorting and Searching Algorithms," *International Research Journal of Engineering and Technology (IRJET)*, vol. 5, no. 1, pp. 1412–1416, 2018.

[2] K. K. Pandey and N. Pradhan, "A Comparison and Selection on Basic Type of Searching Algorithm in Data Structure A Comparison

and Selection on Basic Type of Searching Algorithm in Data Structure," *International Journal of Computer Science and Mobile Computing (IJCSMC)*, vol. 3, no. 7, pp. 751 – 758, 2014.

[3] B. Subbarayudu, L. L. Gayatri, P. S. Nidhi, P. Ramesh, R. G. Reddy, and K. K. R. C, "Comparative Analysis on Sorting and Searching Algorithms," *International Journal of Civil Engineering and Technology (IJCIET)*, vol. 8, no. 8, pp. 955–978, 2017.

[4] V. P. Parmar, "Comparing Linear Search and Binary Search Algorithms to Search an Element from a Linear List Implemented through Static Array , Dynamic Array and Linked List," *International Journal of Computer Applications*, vol. 121, no. 3, pp. 13–17, 2015.

[5] R. Rahim, S. Nurarif, M. Ramadhan, S. Aisyah, dan W. Purba, "Comparison Searching Process of Linear, Binary and Interpolation Algorithm Comparison Searching Process of Linear, Binary and Interpolation Algorithm," *Journal of Physics: Conference Series*, vol. 930, no. 1, p. 012007, 2017.

[6] A. Kumari, R. Tripathi, M. Pal, and S. Chakraborty, "Linear search versus binary search: a statistical comparison for binomial inputs," *International Journal of Computer Science, Engineering and Applications (IJCSEA)*, vol. 2, no. 2, pp. 29–39, 2012.

[7] G. B. Balogun, "A Modified Linear Search Algorithm," *African Journal of Computing & ICT*, vol. 12, no. 2, pp. 43–54, 2019.

[8] S. Tirumala and P. Srinivas, "A new iterative linear programming approach to find optimal protective relay settings," *International Transactions on Electrical Energy Systems*, vol. 31, no. 1, pp. 1–20, 2020, doi: 10.1002/2050-7038.12639.

[9] M. Drmota and W. Szpankowski, "A Master Theorem for Discrete Divide and Conquer Recurrences," *Journal of the ACM (JACM)*, vol. 60, no. 3, pp. 1–49, 2013.

[10] A. E. Jacob, N. Ashodariya, and A. Dhongade, "Hybrid Search Algorithm Combined Combined Linear and Binary Search Algorithm," *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, pp. 1543–1547, 2017.

[11] W. M. Kunkle and P. S. Harrisburg, "The Impact of Different Teaching Approaches and Languages," *ACM Transactions on Computing Education*, vol. 16, no. 1, pp. 1–26, 2016.

[12] R. Mccauley, B. Hanks, S. Fitzgerald, and L. Murphy, "Recursion vs . Iteration : An Empirical Study of Comprehension Revisited," *Proceedings of the 46th ACM technical symposium on computer science education*, pp. 350–355, 2015.

[13] C. Mirolo, "Is Iteration Really Easier to Learn than Recursion for CS1 Students?," *Proceedings of the ninth annual international conference on International computing education research*, pp. 99–104, 2012.

[14] C. Rinderknecht, "A Survey on Teaching and Learning Recursive Programming," *Informatics in Education-An International Journal*, vol. 13, no. 1, pp. 87–119, 2014.

[15] H. Zhang, B. Zhao, W. Li, Z. Ma, and F. Yu, "Resource-Efficient Parallel Tree-Based Join Architecture on FPGA," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 66, no. 1, pp. 111–115, 2018, doi: 10.1109/TCSII.2018.2836920.

[16] E. Lutfina, F. L. Ramadhan, U. N. Karangturi, and K. Semarang, "Perbandingan Kinerja Metode Iteratif dan Metode Rekursif dalam Algoritma Binary Search," *SEMINAR NASIONAL APTIKOM (SEMNASTIK) 2019*, pp. 53–60, 2019.